Parallel Direct Simulation Monte Carlo Computation Using CUDA on GPUs

C.-C. Su[a], C.-W. Hsieh[b], M. R. Smith[b], M. C. Jermy[c] and J.-S. Wu[a]

[a]*Department of Mechanical Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu 30010, TAIWAN*
[b]*National Center for High-Performance Computing, 7 R&D Road, Hsinchu 30076, TAIWAN*
[c]*Department of Mechanical Engineering, University of Canterbury, Private Bag 4800, Christchurch 8140, New Zealand*

Abstract. In this study computations of the two-dimensional Direct Simulation Monte Carlo (DSMC) method using Graphics Processing Units (GPUs) are presented. An all-device (GPU) computational approach is adopted-where the entire computation is performed on the GPU device, leaving the CPU idle-which includes particle moving, indexing, collisions between particles and state sampling. The subsequent application to GPU computation requires various changes to the original DSMC method to ensure efficient performance on the GPU device. Communications between the host (CPU) and device (GPU) occur only during problem initialization and simulation conclusion when results are only copied from the device to the host. Several multi-dimensional benchmark tests are employed to demonstrate the correctness of the DSMC implementation. We demonstrate here the application of DSMC using a single-GPU, with speedups of 3~10 times as compared to a high-end Intel CPU (Intel Xeon X5472) depending upon the size and the level of rarefaction encountered in the simulation.

Keywords: Rarefied Gas Dynamics, Parallel Simulation, Direct Simulation Monte Carlo (DSMC), CUDA, GPU.
PACS: 47.45-n ; 47.45.Ab ; 52.65.Pp

INTRODUCTION

The Direct Simulation Monte Carlo (DSMC) method is a computational tool for simulating flows in which effects at the molecular scale become significant [1]. The Boltzmann equation, which is appropriate for modeling these rarefied flows, is extremely difficult to solve numerically due to its high dimensionality and the complexity of the collision term. DSMC provides a particle based alternative for obtaining realistic numerical solutions. In DSMC the movement and collision behavior of a large number of representative “simulation particles” within the flow field are decoupled over a time step which is a small fraction of the local mean collision time. The computational domain itself is divided into either a structured or unstructured grid of cells which are then used to select particles for collisions on a probabilistic basis and also are used for sampling the macroscopic flow properties. The method has been shown to provide a solution to the Boltzmann equation when the number of simulated particles is large enough [2]. The sizes of DSMC cells have to be much smaller than the local mean free path for a meaningful simulation in general.

NUMERICAL METHOD

Since its introduction, the Direct Simulation Monte Carlo (DSMC) [1] has become the standard method for simulating rarefied gas dynamics. It is generally very computationally intensive, especially in the near-continuum (collision-dominated) regime. The general wisdom for accelerating the DMSC computation is to parallelize the code using the MPI protocol running on clusters with large numbers of processors such as PDSC by Wu *et al.* [3]. Such implementations rely upon the Multiple Instructions on Multiple Data (MIMD) parallelization philosophy and parallel efficiency over massive numbers of nodes is not optimal. Recently, Graphics Processing Units (GPUs) have become an alternative platform for parallelization, employing a Single Instruction on Multiple Data sets (SIMD)

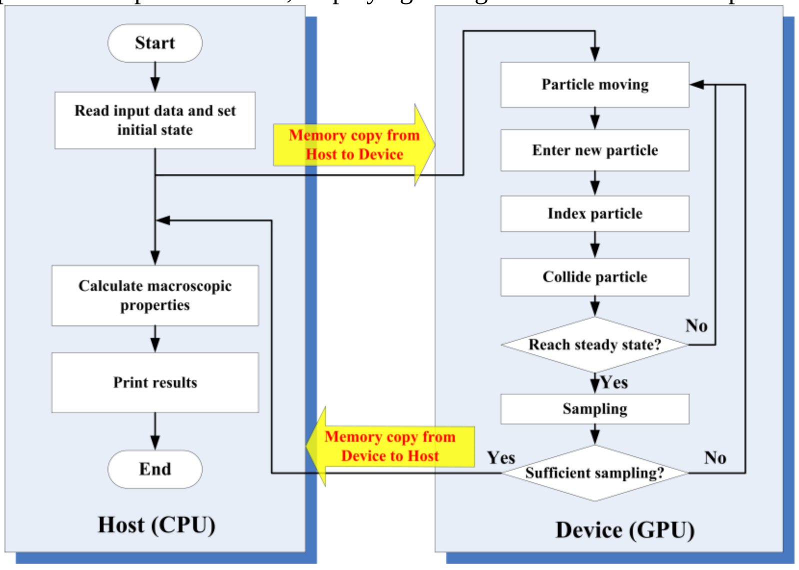

FIGURE 1. A flowchart describing the application of DSMC to GPU-accelerated computation.

parallelization philosophy. The resulting parallelization is much more efficient at the cost of flexibility – as a result, the computational time of several scientific computations, especially those which are optimally applied to vectorized computation strategies, have been demonstrated to reduce significantly. The application of GPU computation also has significant advantages in lower power consumption and significantly reduced equipment costs.

Experience showed that the higher the locality of the numerical scheme/algorithm, the higher the speedup is. However, there seems no successful previous study applying GPUs to accelerate the DSMC computation, which employs a (generally) highly local algorithm. The DSMC method, which is a particle-based method developed by Bird [1], is described in Figure 1. Following initialization, the DSMC method generally involves:

- Moving all the simulated particles, which includes treatment of boundary conditions,
- Indexing all simulation particles (sorting particles into cells),
- Performing collisions between particles – outcomes of collisions are stochastic in nature, hence the use of the term “Monte Carlo” in the DSMC name,
- sampling the molecules within cells to obtain the macroscopic quantities.

In this study, an all-device (GPU) computational approach is adopted, which includes particle moving, indexing, colliding between particles and sampling. This required some changes of the original DSMC method in order to allow efficient all-device computation. Figure 1 shows the flowchart of DSMC computation using a single-GPU. During the initialization stage, input data is loaded into memory and initial states are determined on the host (CPU). This information (including particle and computational cell information) is transferred to the GPU device global memory. Following this, the unsteady phase of the DSMC simulation is performed - particle moving, indexing, particle selection and collisions and sampling are executed on GPU. During particle movement, each particle is tracked by a thread, with each thread tracking $N_p/N_{thread}+1$ particles, where N_p is total simulated particles and N_{thread} is number of threads employed by the GPU device. Each thread reads/writes particle data to/from the global memory of the GPU device [4]. The particle indexing phase of the computation is similar to Bird’s DSMC implementation [1]. We use a function contained within the Software Development Kit (SDK) of CUDA, *scanLargeArray*, to scan through data elements of large arrays contained within global memory. This function is used to efficiently perform particle indexing. During the collision phase, a different parallelization philosophy employed - all particle collisions within a cell are handled by a single thread, allowing efficient recollection of data since all data is coalesced. During the sampling phase, shared memory of GPU [4] is used to store properties of all particles in each cell. After all particles within the same cell are sampled, we copy the sampled data from the (faster) shared to (larger) global memory. When our simulation is nearing completion (i.e. the flow has reached steady state and the sampled data is

TABLE 1. Simulation conditions for the supersonic flow over a horizontal flat plate benchmark.

	Ma	Kn	Number density (#/m^3)	Number of cells	# particles
Case I	4.4	0.0143	1.E20	200x120	~1,320,000
Case II	4.4	0.00715	2.E20	400x240	~5,120,000
Case III	4.4	0.003575	4.E20	800x480	~20,000,000

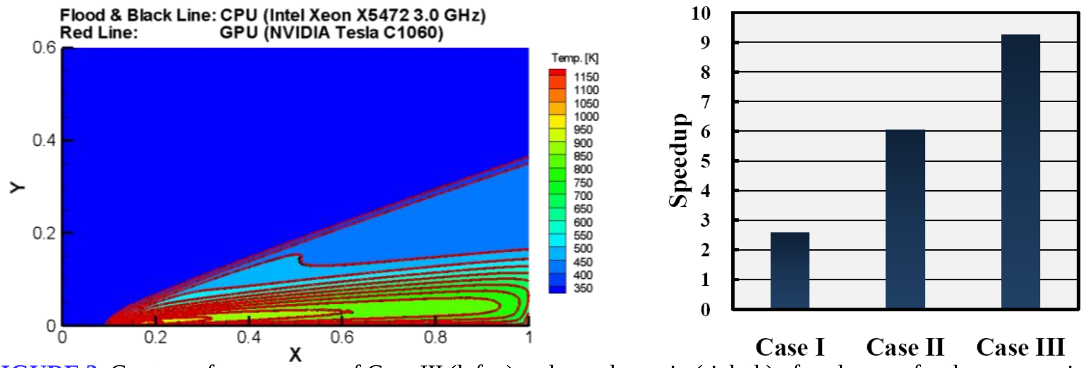

FIGURE 2. Contour of temperature of Case III (left a) and speedup ratio (right b) of each cases for the supersonic horizontal flat plate problem.

sufficient to remove undesired statistical scatter), we move the sampled data from device (GPU) global memory to CPU memory. Finally, calculation of the macroscopic properties is performed by the host and the data is written to file for further analysis.

RESULTS AND DISCUSSION

In this study, we verify our DSMC implementation and demonstrate significant speedup of DSMC computations using a single GPU device through the simulation of several two dimensional benchmark problems: namely, (i) supersonic flow over a (fixed temperature) horizontal flat plate, and (ii) a supersonic lid-driven cavity problem. Approximately 50 particles per cell for all benchmark test cases are maintained throughout the simulations. The DSMC computation employs a VHS collision model [1] for each case. All benchmark simulations are performed on the latest high end computation equipment: single CPU computations employ an Intel Xeon X5472 CPU (3.0 GHz, 12 MB Cache) while GPU computations employ an Nvidia Tesla C1060 (240 microprocessors @ 1.4 GHz, 4 GB DDR3 global memory) hosted by the same Intel Xeon CPU employed for the single CPU test cases.

Two-Dimensional Supersonic Flow over a Horizontal Flat Plate

This benchmark involves the two-dimensional supersonic flow over a flat plate. Ideal argon ($\gamma = 5/3$) with temperature 300 K is initially assumed to be moving with Mach number M = 4.4 over a diffusely reflecting flat plate of fixed temperature 500 K. The length of the flat plate is L = 0.9m, with the initial flow-field density based on the Knudsen number (computed with characteristic length based on the plate length). The initial simulation conditions are summarized in Table 1.

Following the initialization, the simulation is allowed to progress in an unsteady fashion until a steady solution is reached, after which samples are taken to eliminate statistical scatter. Figure 2a shows the resulting contours of temperature for Case III. The results show that the GPU code can reproduce the data simulated by the serial (CPU) code with allowances for statistical scatter. Figure 2b shows the speedup obtained using GPU as compared to using a single core of the Intel Xeon X5472 CPU for each case. We demonstrate a decrease in computational time of 3-10 times when the GPU device specified is employed for the simulation.

Table 2 summarizes the computational time and speedup of each component of all cases using both CPU and GPU. We observe that the speedup using GPU computing increases with reduced rarefaction of the flow. This is also justified since (i) application of a GPU device requires significant overhead due to data transfer and device initialization, and (ii) the general ratio of memory bound communications to device computations is reduced for larger flow problems, resulting in more time spent in computation than in communication. In the current implementation, the sampling phase of the simulation performs best due to the ideal parallelization using the device shared memory.

TABLE 2. Computational time (unit: second) and speedup ratio using a single CPU (Intel Xeon X5472) and single-GPU device (Nvidia Tesla C1060) for the supersonic horizontal flat plate problem.

	Case I			Case II			Case III		
	CPU	**GPU**	**Speedup**	**CPU**	**GPU**	**Speedup**	**CPU**	**GPU**	**Speedup**
Total	18456.5	7175.95	2.57	108721	17966.4	6.05	602674	65108.9	9.26
Particle moving	8271.31	5325.07	1.55	56100.3	10754.9	5.22	308995	30933.6	9.99
Index	3612.49	1486.69	2.43	25310.3	5839.58	4.33	168463	26894.1	6.26
Collision	1884.41	240.407	7.84	7611.54	889.106	8.56	33433.1	3490.77	9.58
Sampling	4680.56	112.332	41.67	19669.5	447.008	44.00	91664.9	3661.53	25.03

TABLE 3. Initial conditions for the supersonic lid-driven cavity problem.

	Ma	**Kn**	**Number density (#/m^3)**	**Number of cells**	**Total particles**
Case I	2	0.01	1.294E20	100x100	~500,000
Case II	2	0.05	2.59E20	200x200	~2,000,000
Case III	2	0.002	6.47E20	500x500	~12,500,000

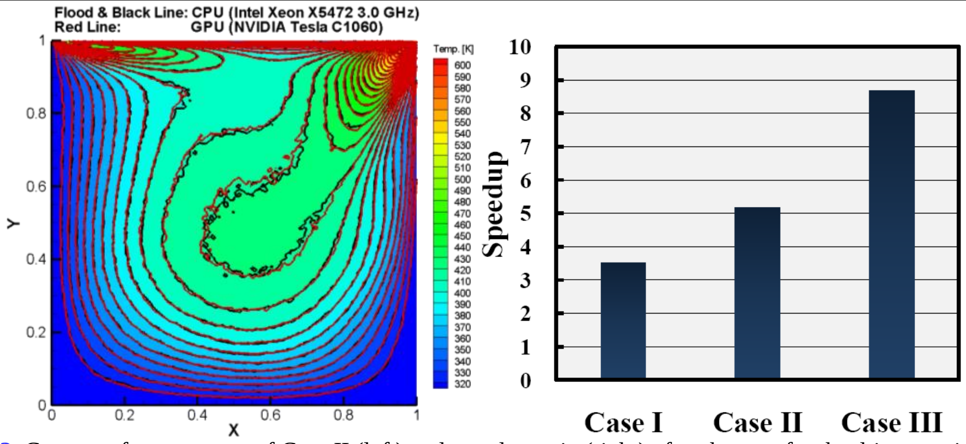

FIGURE 3. Contour of temperature of Case II (left) and speedup ratio (right) of each cases for the driven cavity problem.

Two-Dimensional Supersonic Lid-driven Cavity Problem

The second test case is a two-dimensional supersonic lid driven cavity problem. Here, the simulation domain is a square cavity (1x1 m) with diffusely reflecting walls of fixed temperature (300 K). All walls are stationary except the upper wall which is moving (with positive velocity) at Mach Number M = 2. The gas (ideal argon, $\gamma = 5/3$) is initially at rest with a temperature of 300 K and various density depending on the governing Knudsen number, computed with a characteristic length equal to the box width. These initial conditions are summarized in Table 3.

Following initialization, the simulation is progressed in time until the flow is steady and samples are taken to reduce the statistical scatter. Figure 3a show the temperature contour of Case II. It is worth noting that the region in the central region of the lid-driven cavity is rarefied, with a low density and relatively high temperature (as also shown in Figure 3). The GPU results are almost identical to the equivalent CPU results which again validate the CUDA GPU implementation. Differences between the results are (probably) able to be explained by differences between the random numbers employed by the GPU and CPU solvers and general levels of statistical scatter. The random numbers employed by the GPU DSMC implementation are drawn from pre-computed (fixed-length) arrays to improve the efficiency of random number use on the GPU device.

Figure 3b shows the speedup using GPU is about 3~9 times as compared to that using a single core of the CPU described above. All computational timings are summarized in the Table 4 for reference. The ratio of communication to computation, a critical factor in GPU efficiency, is minimal for larger DSMC problems. Hence, the speedup using GPU computing is shown to increase with reducing rarefaction of the flow (i.e. increasing collision dominance). The sampling phase of the DSMC simulation is shown to be the most efficient phase of the computation when applied to GPU computation due to the low number of communications required.

TABLE 4. Computational time (unit: second) and speedup using CPU of Intel (Xeon X5472) and single-GPU of NVIDIA (Tesla C1060) for the driven cavity problem.

	Case I			**Case II**			**Case III**		
	CPU	**GPU**	**Speedup**	**CPU**	**GPU**	**Speedup**	**CPU**	**GPU**	**Speedup**
Total	5418.25	1539.33	3.52	28371.2	5489.51	5.17	262779	30276.4	8.68
Moving	1332.38	372.203	3.58	10671	1496.5	7.13	113056	9929.99	11.39
Indexing	1286.98	477.102	2.70	5227.19	1847	2.83	58920.1	11539.4	5.11
Collisions	1580.59	653.678	2.42	6917.35	1998.7	3.46	51990	7754.27	6.70
Sampling	1214.9	31.8918	38.09	5545.75	136.489	40.63	38753.2	986.009	39.30

CONCLUSION

Presented here is the application of DSMC to a novel (entire-device based) GPU acceleration. In the implementation discussed, the CPU is only employed during the initialization and concluding phases of the simulation – the main processes employed during the DSMC simulation are performed on the GPU device while the CPU remains idle. The resulting computations demonstrate a speedup of 3-10 times, depending on problem size, resulting from the lack of communication-bound processes. Efficient use of various memory (caches) on the GPU device for different phases of the DSMC simulation also allow high levels of parallelization. The GPU-DSMC code has been verified through comparison against a conventional serial DSMC computation, with results showing excellent comparison in a fraction of the time. The GPU device used for the benchmark problems (Nvidia Tesla C1060) is commonly available for a fraction of the price associated with a conventional computer cluster required to match its performance.

ACKNOWLEDGMENTS

We would like to thank the National Center for High-Performance Computing (NCHC) of TAIWAN for support through access to the high-performance GPU cluster. Prof. J.-S. Wu would like to acknowledge support from the National Science Council (NSC) of TAIWAN through Grant NSC-99-2922-I-009-110.

REFERENCES

1. G.A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Oxford University Press, 1994.
2. W. Wagner, *J. Stat. Phys.* **66**, 1011-1044 (1992).
3. J.-S. Wu and Y.-Y. Lian, *Computers & Fluids*, **32**, 1133-1160 (2003).
4. NVIDIA Inc., *CUDA Programming Guide Version 2.3.1.*, 2009.