PARALLELIZATION OF DSMC-ALGORITHM ON GPU

DUSHYANT SINGH UDAWAT, ANKIT BANSAL[2]
[1]*Undergraduate, Mechanical and Industrial Department, IIT Roorkee*
[2]*Assistant Professor, Mechanical and Industrial Department, IIT Roorkee*

Abstract: The Direct Simulation Monte-Carlo (DSMC) is a widely used particle-based method to simulate gas flows in rarefied gas regime. It allows molecular movement and collision phase to be decoupled. This gives DSMC lot of potential for parallelization. The DSMC solver already allows parallelization using domain decomposition approach, where entire domain is decomposed into smaller domains and each domain is assigned to a different processor. With a large number of cores (up to 2500 cores), cell based calculations and individual particle tracking can be done much more efficiently on a graphical Processing Unit (GPU). GPU parallelization is implemented at the level of molecular movement and inter-molecular collisions in existing DSMC solver. The accuracy, efficiency and feasibility of the GPU implementation in DSMC is analysed and presented in this paper. The paper also suggests that a more robust simulation would the one in which the whole processing is done on GPU and only the final output is copied to the host memory.

Keywords: DSMC; GPU; CUDA; Parallelization; Lagrangian Approach; Unified Memory

INTRODUCTION

In continuum flow regime, flows can be approximated by the use of Navier-stokes equation by using the Finite Element method. The finite element method proves to be very expensive method for the flow simulations in the case when there are very limited no. of molecules in a cell of the mesh. Direct Simulation Monte Carlo method is used in some situation. Whether a flow satisfies the continuum hypothesis or not is found by Knudsen number.

$$Kn = \frac{\lambda}{L}$$

λ, mean free path of gas molecules

L, characteristic dimension of flow

With smaller values of Kn(<0.01), the Navier-stokes equation predicts the result accurately but for larger values(Kn > 0.1), Navier-stokes equation become increasingly inaccurate. For this reason, a lagrangian method is proposed, in which the particles are followed and not the position. This methodology is very realistically applied in the simulation as various models are available for probabilistic collisions and interaction between gas molecules. This eliminates the need for approximation that would otherwise have been a necessity in the Navier-stokes equations. The collision interaction model used in this paper is the Larsen-Borgnakke(LBS) Method and the Variable Hard Sphere(LHS). If the collisions between the gas molecules happen with sufficient energy between them then it may result in various kinds of reactions between the gas molecules. These reactions include recombination, dissociation, chain and exchange reaction. For simplicity, we are limiting the scope of this paper to non-reacting

*Author for Correspondence: E-mail: abansfme@iitr.ac.in Tel.: +91-01332284781

mixtures only. Also, the DSMC algorithm has been extended for 2D and 3D flows. For simplicity and demonstration of speed-up, only 1D homogenous gas flow is considered. The effect of parallelization will be larger in the case of 2D and 3D.

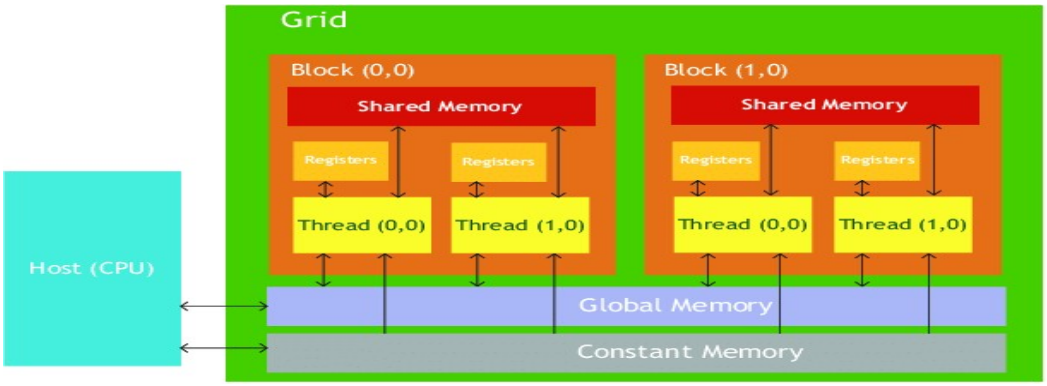

Figure 1 Memory architecture

GPU AND CUDA ARCHITECTURE

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by NVIDIA. The architecture allows the software developer to harness the capabilities of the graphic processing unit provided by NVIDIA. The memory hierarchy is shown in figure below. The figure explains the interaction between the GPU memory and the CPU memory and their sub-categories.

The memory typed in the GPU are-

1. Global memory
   This is the biggest memory in the GPU. The host memory can transfer or receive data from Global Memory.
2. Shared Memory(SM)
   It is a fast access memory accessible inside a block.
3. L1 cache
   Cache memory accessible from a block.
4. Local memory
   Local memory is only accessible in a thread and is as fast as SM.

The thread and the block architecture is depicted in the figure below.

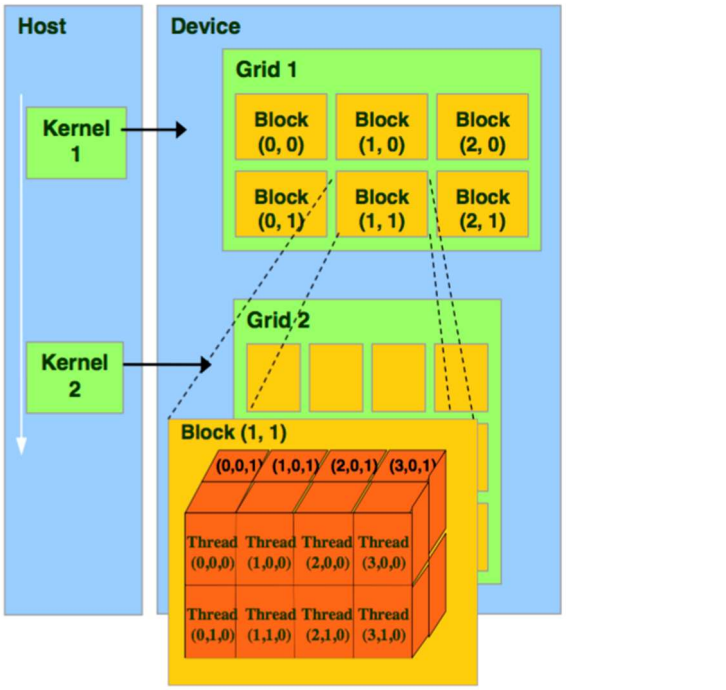

Figure 2 Grid Architecture

It is clear that the host gives commands for the blocks to start executing. Every block has several threads (1024 maximum). And every grid consists of many blocks. A kernel is a specific number of blocks executing in parallel, performing a single function. CUDA architecture in the following paper do not allow simultaneous kernels.

DSMC ALGORITHM

The flowchart of typical DSMC programs has not changed greatly since the method was first introduced, but the modern features have

necessitated some modifications. The molecule moves and collisions are uncoupled over a small time step that is set to a fixed fraction of the local mean collision time. The density in some applications varies by several orders of magnitude over the flow-field. The region of highest density is often the most important and a good calculation requires that the time step be small in comparison with the mean collision time in that region. An older program with a fixed time step is then very inefficient and a variable time step is now employed. However, there is still some penalty because the main loop must be over the minimum time step. This appears in the green process block in the flow chart, while the local time steps that may be very much larger appear in the orange process blocks. Note that the molecules move and collisions are calculated in a collision cell when their times fall more than half a time step behind the overall time step. This means that the average molecule and collision times are, at all times, equal to the flow time.

The time step was a data item in the early programs, but is now set automatically by the program. It is initially set to the value appropriate to the stream or reference gas and is updated to the current flow conditions whenever results are output. The sampling intervals are set as multiples of the time step and the output interval to multiples of the sampling intervals. The multiples are set to default values that may be altered through the adjustable computational parameters in the source code. The only computational parameter in the data files is the initial number of megabytes to be used in the calculation, but the computational parameters within the source code allow users to make adjustments that hopefully lead to optimal settings for special cases.

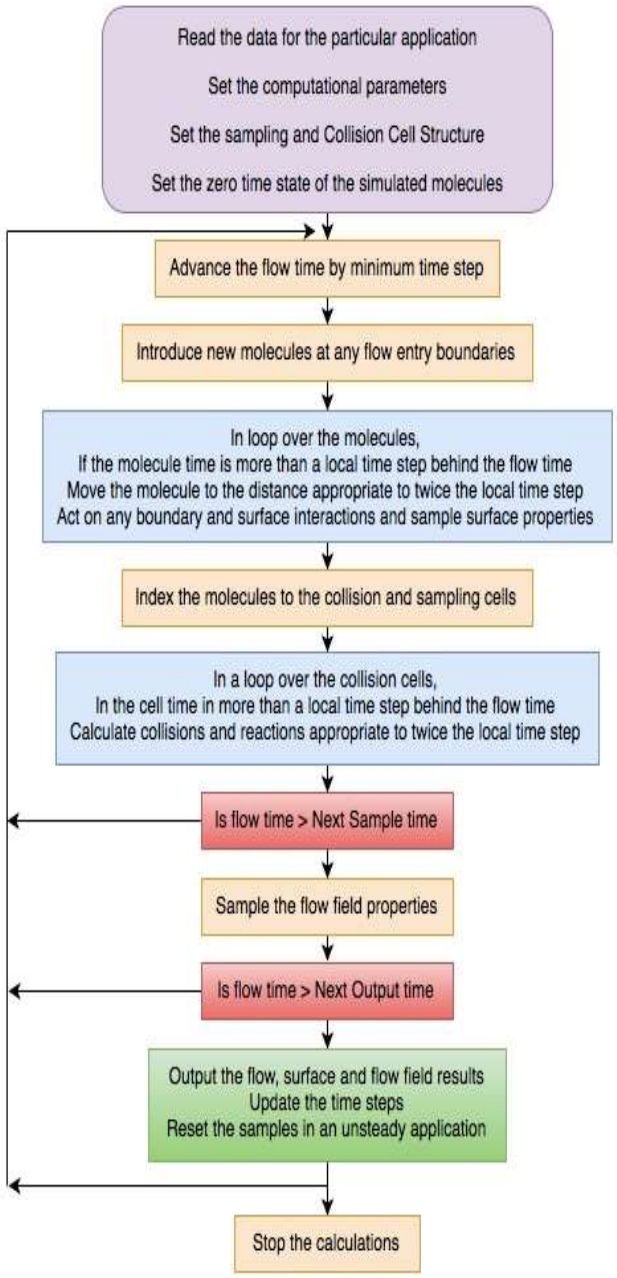

Figure 3 Flowchart of the DSMC Method

The method has been divided into the following main iterative steps –

1. Molecules_enter (for adding the molecules in the flow domain)
2. move_mols (for moving the molecules)
3. index_mols (for molecules indexing)
4. collision_mols (for collision of molecules in collision cells)
5. sample_cells (sampling of data for all the sampling cells)

GPU IMPLEMENTATION

1. Data structure
   The data related to the molecules are stored in different arrays. There are separate arrays for velocity, position, temperature etc. such that i-th element of the array represents the i-th molecule of the flow domain. An alternative method would have been a class where all the data related to a single molecule can be stored. But, it turns out from the experiment that this kind of structure can't be efficiently processed on the GPU, thereby decreasing the bandwidth of the computation. The 1D arrays can be directly, conveniently processed by a GPU and with much more speed.

2. GPU Kernels
   During the indexing of molecules, all the molecules are assigned the collision cell (CC) number and the sampling cell (SC) number. The CC is used to complete the collisions between the molecules while the SC are used to sample the outputs. As already mentioned, the papers focuses on demonstrating the speed-up gained through move_mols() and the collision_mols() functions. So, parallelization of the code takes place here only. In move_mols function, all the molecules undergo their independent movement, so the number of kernels launched is equal the number of molecules in the flow domain. These threads run in parallel to produce speed-ups in the program. However in the collision_mols function, there exist serious interaction between the molecules and therefore separate molecules can't run in parallel. For this reason, collision cells are used in accordance with the Larsen-Borgnakke method. A certain number of random collisions between molecules takes place inside the collision cells, changing the velocity and energy in the molecules. Due to this reason, number of kernels launched here will be equal to the number of CC.

RESULTS AND DISCUSSIONS

The Results of the experiment are shown in Table 1 below. A speed-up of 1.8x has been achieved in the move_mols function while a speed-up of 2.8x has been achieved in the collision_mols function. This should be noted that as the time taken by these two function keeps decreasing the need to accelerate the index_mols function increases as the index_mols function becomes the new bottleneck of the program. Also, there is a great need to parallelize the sample_mols function. Some of the challenges faced during the implementation of the parallezied code are as follows –

a. Unified Memory
   Unified memory has been used to store most of the data on the GPU.
b. Linearization of arrays
   In the original fortran code the arrays used are of multiple dimension (2D, 3D, 4D and 5D) but this causes a great deal of inefficiency during processing. This leads to slowed speeds, the code was edited and teach multidimensional arrays was converted into a 1D array. All the dat5a of the former array is to be stored in the new arrays.

Function Name	Execution time	Execution time after parallelization	Speed-up achieved
Move_mols	0.02677 ms	0.015155 ms	1.8x
Collision_mols	0.03435 ms	0.01236 ms	2.8x

c. Storing of array sizes into the shared memory
   The linearized arrays poses a serious problem since the sizes of various sections are stored inside the arrays only then it becomes expensive to extract information as one call would lead to 3 to 5 extractions from the array. However, if the array sizes are stored in the shared memory then the access to array's data becomes much faster.
d. Storing of fixed arrays in the shared memory
   The arrays whose values do not change has been stored in the shared memory for faster access. Also, data which will be accessed very frequently has been stored in the local memory of GPU

CONCLUSION

GPU parallel computing is implemented on DSMC Code. With only move molecule and collision stage parallelized, The program has achieved a speed-up of approximately 1.8x and 2.8x respectively for the stages, when compared to total collision time taken by single core processor of Intel® Xeon(R) ES-2670 v2 CPU. To produce these results 512 cores of NVIDA Quadro K4000 are used. The index_mols and sample_mols function also needs to be parallelized as they are now the bottleneck of the program. The overhead due to the use of unified memory also emphasises on the need to parallelize all the iterative functions. These results are very promising, and as discussed above, it can be used to make fully GPU parallelized DSMC solver.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

REFERENCES

Bird GA. (1994) Molecular gas dynamics and the direct simulation of gas flows *Clarendon Press* Oxford

Dietrich S, Boyd I 1996 Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method *Journal of Computational Physics* **2** 328-342

Goldsworthy M.J. 2014 A GPU–CUDA based direct simulation Monte Carlo algorithm for real gas flows *Computers & Fluids* ***94*** 58-68

Greenshields C J (2015) OpenFOAM: The open source CFD Toolbox, User Guide version 3.0.1 *OpenFOAM Foundation Ltd.*

Huang Fei, Jin Xuhong, Mei Wenlong, Cheng Xiaoli 2015 A new parallel algorithm of dsmc method *Proceeding of 7th International Conference on Fluid Mechanics, ICFM7* 622-627

Ivanov M S, Markelov G N, Gimelshein S F 1998 Statistical simulation of reactive rarefied flows – Numerical approach and application *7th AIAA/ASME Joint Thermophysics and Heat Transfer Conference (AIAA Paper 98)* 2669 Albuquerque, NM, U.S.A. June 1998

Message Passing Interface Forum (2015) MPI: A Message-Passing Interface Standard Version 3.1 *University of Tennessee, Knoxville, Tennessee (http://www.mpi-forum.org/)*

NVIDIA Corporation (2015a) NVIDIA CUDA C Programming Guide v7.5 *NVIDIA*

NVIDIA Corporation (2015b) NVIDIA CURAND Programming Guide v7.5 *NVIDIA*

Padilla JF, Boyd I 2009 Assessment of Gas-Surface Interaction Models for Computation of Rarefied Hypersonic Flow *Journal of Thermophysics and Heat Transfer* **23**, No. 1 96-105

Plimpton S, Gallis M (2016) SPARTA Documentation *US Department of Energy (http://sparta.sandia.gov/)*

Shen Ching (2006) Rarefied Gas Dynamics: Fundamentals, Simulations, and Micro Flows *Springer Science & Business Media*

Scanlon T J , Roohi E, White C, Darbandi M, Reese J M 2010 An open source, parallel DSMC code for rarefied gas flows in arbitrary geometries *Computer & Fluids* ***39*** 2078-2089

Shang Zhi, Chen Shuo 2013 3D DSMC Simulation of Rarefied Gas Flows around a Space Crew Capsule Using OpenFOAM *Open Journal of Applied Sciences* **3** 35-38

Su C.-C., Smith M.R., Kuo F.-A., Wu J.-S., Hsieh C.-W., Tseng K.-C. 2012 Large-scale simulations on multiple Graphics Processing Units (GPUs) for the direct simulation Monte Carlo method *Journal of Computational Physics* **231** 7932-7958