KINETIC MODELING AND SIMULATION OF HYPERSONIC, SHOCK-BOUNDARY LAYER INTERACTIONS USING PETASCALE COMPUTING

Allocation: Illinois/70.0 Knh
PI: Deborah A. Levin[1]

[1]University of Illinois at Urbana Champaign

EXECUTIVE SUMMARY

The modeling of shock-boundary layer interactions of three-dimensional hypersonic flows using kinetic, particle methods such as direct simulation Monte Carlo (DSMC), provides the highest fidelity in understanding thermochemical non-equilibrium processes when extremely complex shock interactions are present. We have developed an Octree based (message passing interface) MPI-parallelized code that takes advantage of Adaptive Mesh Refinement (AMR) techniques to maximize the placement of computational particles in flow regions where the collision frequency is highest. Such continuum-like conditions create massive computational loads in simulating several billions of particles. To perform such continuum-like computations, it was important that the code have superior algorithmic efficiency amenable to high scalability. This code required the implementation of Morton-Z space filling curve that facilitates a direct access of leaf cells in an Octree and careful attention to efficient use of the cache.

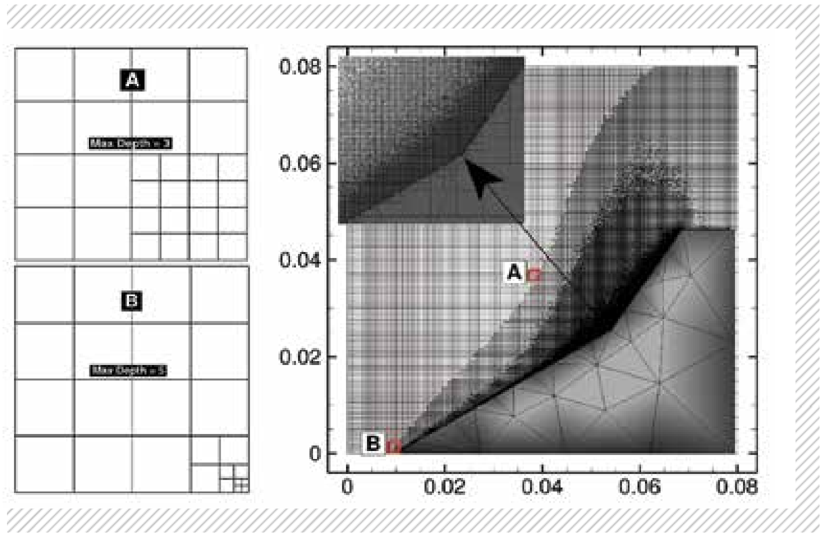


FIGURE 1: Need for linearization for hypersonic modeling of shock-boundary layer interactions.

INTRODUCTION

Hypersonic flow over a double wedge configuration at continuum-like free stream conditions has been a challenging problem because of the multiple shock-shock and shock-boundary layer interaction, separated flows near the hinge, sheer layer, and three-dimensional effects. These conditions generate a mesh that is highly non-uniform because of very high levels of refinement near the wedge as shown in Fig. 1. It can be seen that the dense mesh obtained after the shock just near the hinge for the aforementioned double wedge case is highly non-uniform because of the high gradients. The Octree cells lying near the surface are highly refined as compared to those in the free stream and inside the geometry. Therefore, these cells have to perform a lot of work while others wait idly at the end of each time step, which is unfavorable as the full capacity of all the processors is not utilized and we essentially see a high degree of imbalance of the load among the processors. Furthermore, at high computational loads, the communication time starts to increase.

METHODS & RESULTS

The main tasks in any DSMC code are (1) the movement of particles, (2) their mapping to the correct computational cell after movement, (3) sampling or the calculation of the particle macroscopic properties, and (4) performing binary collisions. Note that; the DSMC algorithm is efficient because it is assumed that particle movement and collisions may be decoupled. The earlier version of our code required recursive access of particle data by linked-lists to accommodate the dynamic changes in the number of particles and the number of computational cells. As a result, it took three times longer, per particle, than a Fortran 77 DSMC code. Upon close examination, it was found that both codes took the same amount of time to perform collisions, but the main difference was due to the recursive pointer based access of cells in an AMR-based unstructured code. To avoid this cost but retain the flexibility of the unstructured mesh, a Morton-Z space filling curve that facilitates direct access to leaf cells in an Octree was implemented. In addition, the entire data structure of the SUGAR code was replaced by arrays instead of a pointer-based system, which now ensured that the code gave superior performance, per processor.

After achieving the optimum algorithmic efficiency along with preserving the unstructured nature of the mesh, the scalability of the code was improved. Figure 2 shows the speed up plot for the strong scaling in comparison with the earlier version of the SUGAR code. The earlier version of the code was partitioned using a simple 2-D blocking algorithm where the division of the domain was done at the root (Octree) level. It was further coupled with the state-of-the-art graph partitioning tool Zoltan in an attempt to obtain better load balancing and reduce the communication between the processors. The speed up curve for a case of the flow of argon over a hemisphere using 2-D blocking algorithm and Scotch graph partitioner is shown in the figure. It can be seen that the maximum speed up by a just factor of four was observed with a 16 times increase in the number of ranks, and the use of Scotch made no difference. However, after many improvements, the current version of the SUGAR code shows a near-ideal speed up by a factor of 64. This increase in speed was a result of the following improvements: 1) The domain was portioned now at the leaf level, now possible with the implementation of space filling curves. This also gave better flexibility in balancing the computational loads. 2) Instead of manually putting the weighting criterion based on the number of particles, cells, or geometry panels in a leaf cell, the code was modified to take into account the time taken to perform each of these tasks after the steady state is reached by profiling these times, it comes up with an optimum weighting criterion during the runtime. 3) Attempts to reduce the communication among neighboring processors by creating a list of possible communication pairs resulted in a tremendous improvement in strong scaling. 4) The introduction of particles into the computational domain at the inlet was parallelized, and efforts for further modifications are ongoing. 5) More efficient use of memory. Regarding weak scaling, the SUGAR code can simulate 3 billion particles with 90% parallel efficiency and the tests for a higher number of processors is ongoing.

WHY BLUE WATERS

The main advantage of Blue Waters is its massive number of nodes. A single actual low Knudsen number double-wedge case takes approximately 10,500 node-hours.

NEXT GENERATION WORK

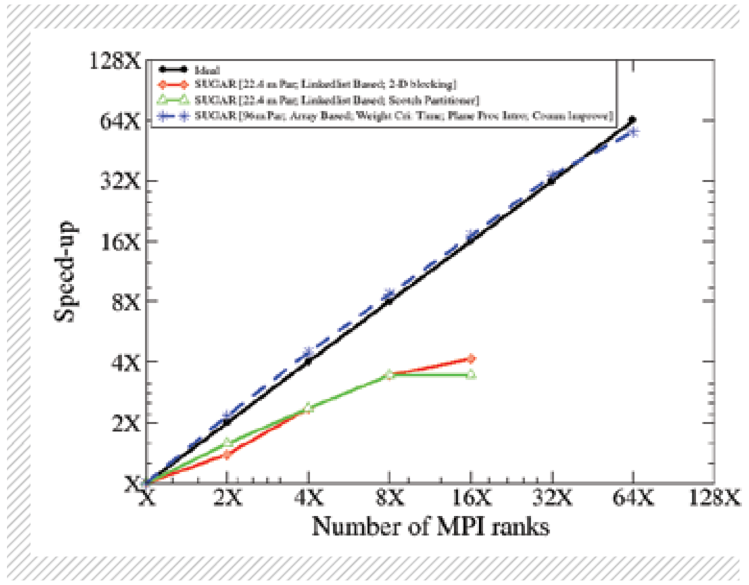
At present, the SUGAR solver can simulate hypersonic neutral and charged particle flows through highly irregular porous geometries and efforts are directed to include chemistry and radiation. We envision new architectures with considerably more memory per CPU to enable us to achieve the levels of parallelization required to solve unsteady, transitional flows.

PUBLICATIONS AND DATA SETS

Sawant, S., R. et al., Study of Shock-Shock interactions Using an Unstructured AMR Octree DSMC Code, AIAA SciTech 2016, January 4th 2016, Paper No. 2016-0501

Jambunathan, R. and D. Levin, Advanced Parallelization Strategies for Modeling Flow Through Ablative Thermal Protection Systems, AIAA Paper No. 2016-3387, (2016), 46th, AIAA Thermophysics Conference, 10.2514/6.2016-3387.

FIGURE 2: Strong scaling performance of the improved SUGAR DSMC code.


Ideal
SUGAR [22.4 m Par; Linkedlist Based; 2-D blocking]
SUGAR [22.4 m Par; Linkedlist Based; Scotch Partitioner]
SUGAR [96 m Par; Array Based; Weight Cri. Time; Plane Proc Intro; Comm Improve]
128X
64X
32X
16X
8X
4X
2X
X
Speed-up
X
2X
4X
8X
16X
32X
64X
128X
Number of MPI ranks