CFDRC • • • Delivering Breakthrough Solutions

Towards GPU Accelerated Parallel Solvers for Multi-Phase Flows with Adaptive Cartesian Mesh

Vladimir Kolobov and Robert Arslanbekov

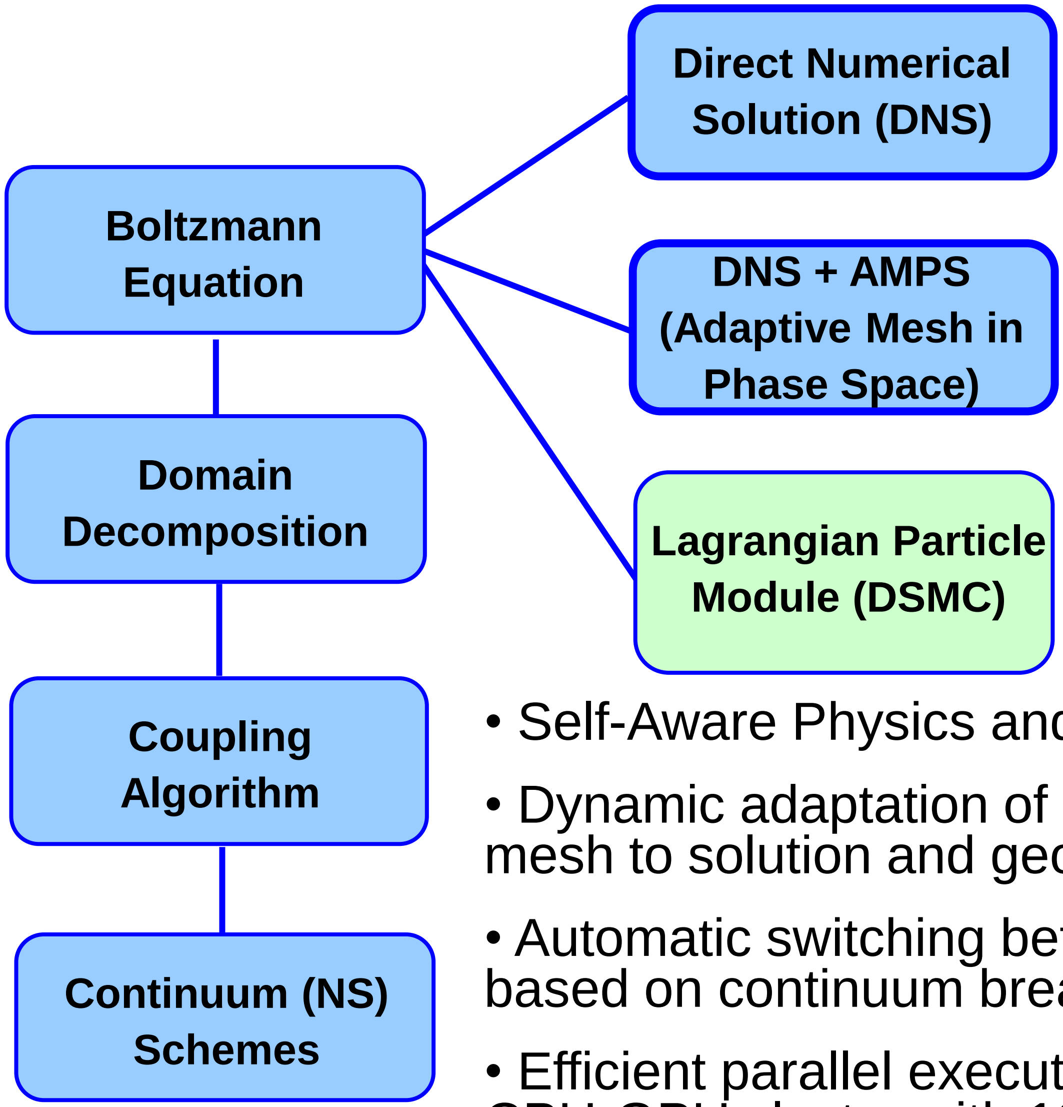
CFD Research Corporation, Huntsville, AL, USA

NETL Workshop on Multiphase Flow Science
September 6-7, 2013

Motivation

- Serial computing has reached its zenith in performance. In the foreseeable future, parallelism will be the key ingredient for increasing performance.
- Most applications benefit from powerful combination of a massively parallel GPU and a fast multicore CPU.
- GPU Architectures: Fermi->Kepler->Maxwell
- Heterogeneous Multi-Core CPU-GPU clusters for HPC
- GPUs are very effective at exploiting parallelism in *regular*, data-parallel algorithms (arrays & matrices operations)
- *Irregular* algorithms arise from complex data structures such as trees and graphs – they are more difficult to parallelize (structured vs unstructured)
- Successful Irregular Computations on GPU

Unified Flow Solver



Boltzmann Equation
Direct Numerical Solution (DNS)
DNS + AMPS (Adaptive Mesh in Phase Space)
Lagrangian Particle Module (DSMC)
Domain Decomposition
Coupling Algorithm
Continuum (NS) Schemes

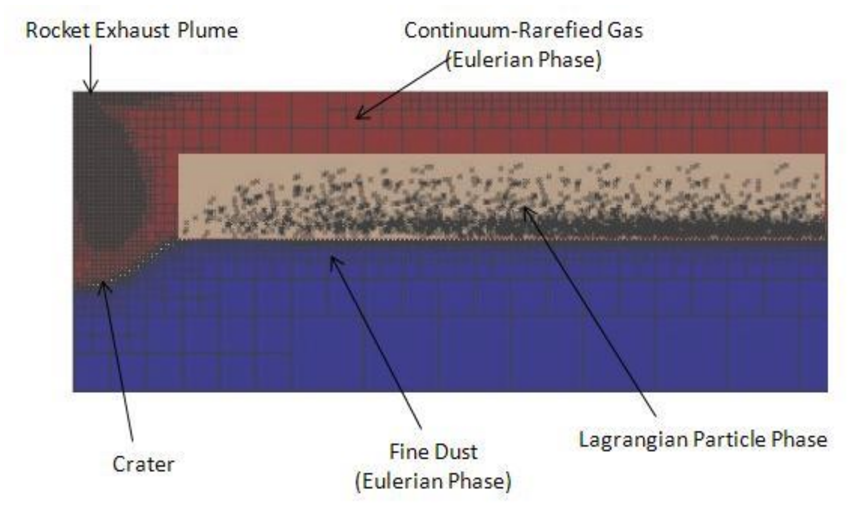
Direct Numerical Solution of the Boltzmann kinetic equation and Particle (DSMC) solvers for coupled atomistic (kinetic) and fluid (hydrodynamic) models with AMAR capabilities

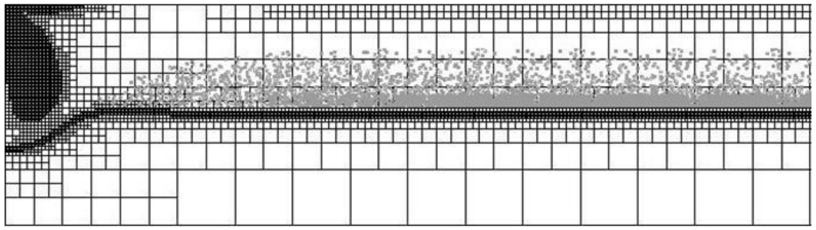
GFS two-phase VoF solver with tree-based AMR

- Self-Aware Physics and Adaptive Numerics
- Dynamic adaptation of computational (Cartesian) mesh to solution and geometry
- Automatic switching between kinetic and fluid models based on continuum breakdown criteria
- Efficient parallel execution (SFC & FoT) on a NASA CPU-GPU cluster with 1000 nodes.

CFDRC Delivering Breakthrough Solutions

Particulate Modules in GFS and UFS


Rocket Exhaust Plume
Continuum-Rarefied Gas (Eulerian Phase)
Crater
Fine Dust (Eulerian Phase)
Lagrangian Particle Phase

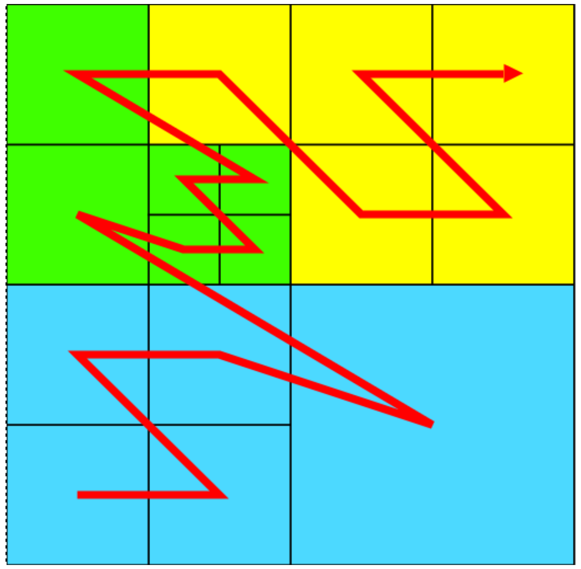


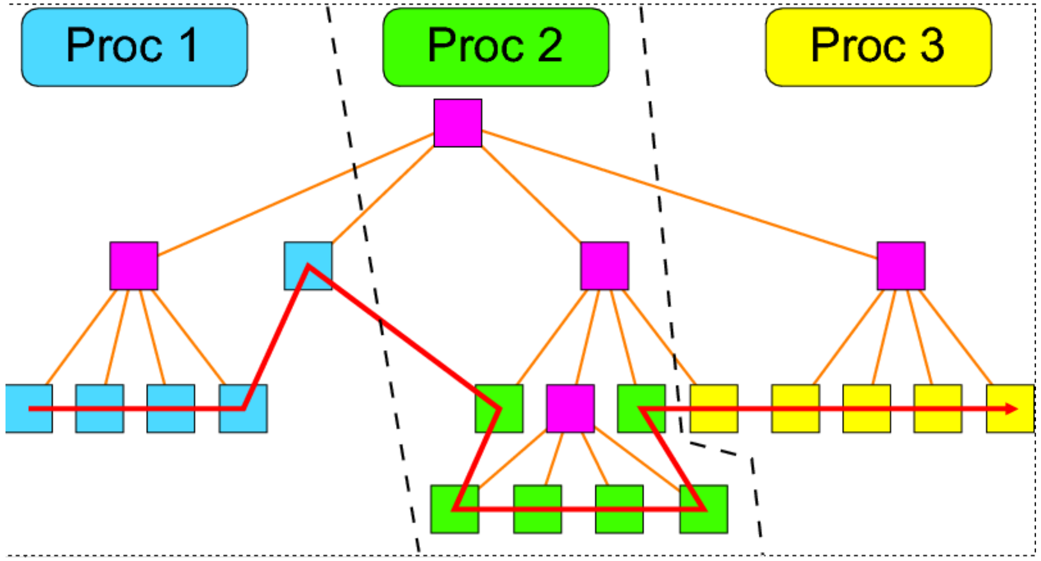
Lagrangian tracking of large particles within the Eulerian-Eulerian gas-dust flow with AMR to flow gradients and to particle density

- **GFS was originally designed for simulations of multi-phase flows using VoF method.**
- **A particulate module was added later for tracking finite-volume solid particles in the Eulerian fluid flow.**
- **The nearest-neighbor search approach has been used with great success in DSMC simulations with adaptive Cartesian mesh achieving extreme parallelism and scaling**
- **This technique has been implemented into the UFS debris transport analysis tool enabling a high-fidelity Eulerian-Lagrangian multi-phase modeling of three component flows: the gas phase, fine dust phase and discrete particle phase.**

Delivering Breakthrough Solutions

Space Filling Curves & Forest of Trees





• **SFC** allows complete flexibility for a fine-grained domain decomposition with highly efficient dynamic load balancing (DLB) among processors.

• During sequential traversing of cells, the physical space is filled with curves in N-order (Morton ordering), and all cells are numbered a one-dimensional array.

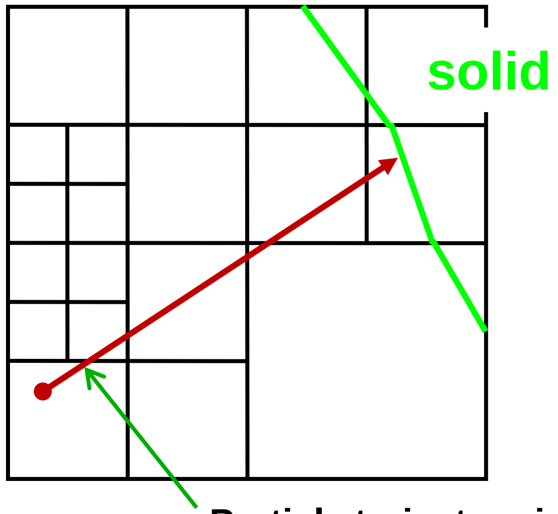
• A weight is assigned to each cell, proportional to CPU time required for computations in this cell. The array modified with corresponding weights, is subdivided into sub-arrays equal to the number of processors.

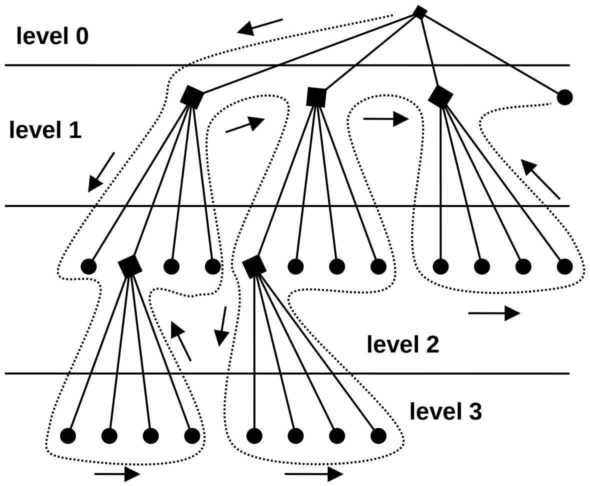
• Coarse-grained domain decomposition is obtained by using multiple octrees (a "**FoT**") connected through their common boundaries. Graph partitioning algorithms are used for domain decomposition and DLB.

UFS-DSMC: Lagrangian Particle Transport Module

- ➢ Use a single, dynamically adapted mesh for (i) particle collision and (ii) statistics collection/visualization and particle movement
- ➢ Tree data structure allows efficient data management for AMR and parallelization of the code
- ➢ DSMC approach requires cell sizes less than local mean free path, λ, → fine grids are necessary in dense flow regions

R.R. Arslanbekov, V. I. Kolobov, J. Burt and E. Josyula, "Direct Simulation Monte Carlo with Octree Cartesian Mesh", AIAA 2012-2990





Particle trajectory involves cells at different levels

Implementation of GPU kernels

- **An all-device (GPU) approach: the entire computation is performed on the GPU device**.
- **Separate kernels for particle moving, indexing, collisions and sampling**
- **Each particle is followed by a separate thread until the particle hits a face of a cell in which it is currently located. At cells faces, particles are either reflected or moved to neighbor cells using neighbor indices.**
- **Particle collisions in each cell are treated by a separate thread.**
- **Sampling of particle locations and velocities are performed with sampling kernels (each cell is treated by a separate thread).**

GPU Kernels are implemented according to

Su C.-C., Smith M. R., Kuo F.-A., Wua J.-S., Hsieh C.-W., and Tseng K.-C., “Large-Scale Simulations on Multiple Graphics Processing Units (GPUs) for the Direct Simulation Monte Carlo Method,” *J. Comp. Phys.* Vol. 231 (2012) 7932-7958.

with modifications for unstructured Cartesian mesh

CFDRC Delivering Breakthrough Solutions

GPU-Accelerated UFS-DSMC: Demonstration

- Intel(R) Xeon(R) X5675 @ 3.07GHz CPU processor and Tesla C2075 GPU device.

- Hypersonic Flow over a cylinder at low Knudsen number (Kn = 0.01) with Mach = 10.

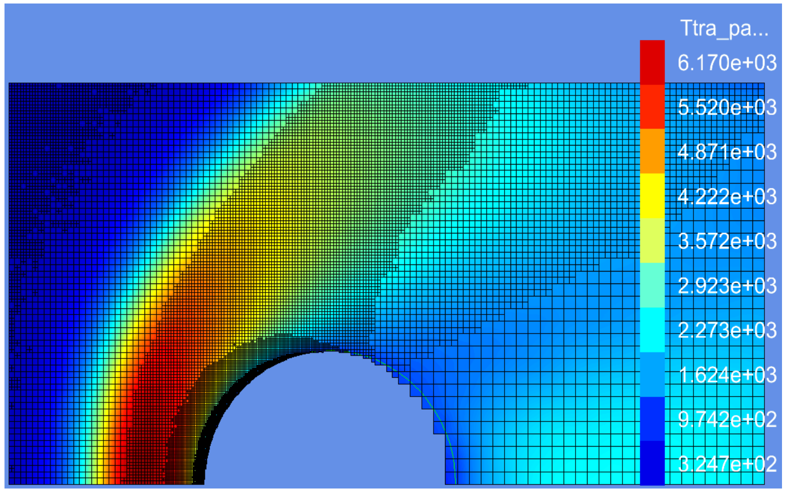
- The free stream temperature was set to 200 K and the wall temperature – to 500 K. A non-uniform grid was used with a finer grid in the stagnation region (denser region) and a coarser grid at the back of the cylinder (rarefied region).

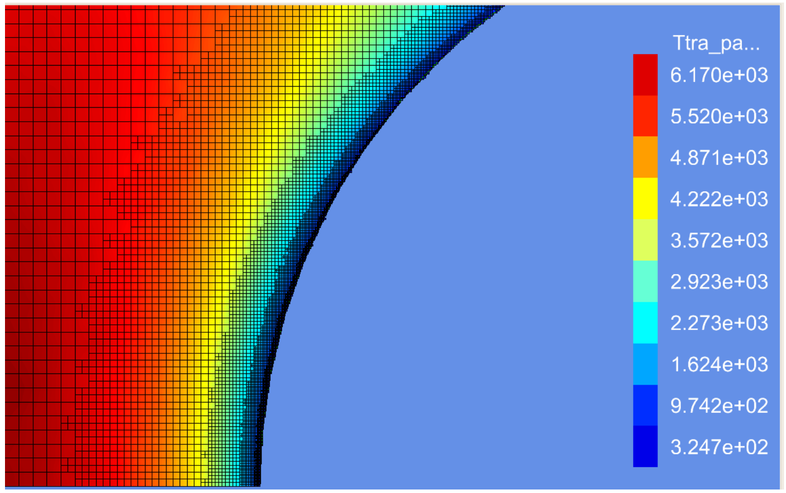
- The number of cells is ~130K for this case. Two cases were benchmarked with the total number of particles of ~0.5M and ~2M

UFS-DSMC: Rarefied Hypersonic Flow

- First 10,000 time steps on a uniform level 5 grid
- Between time steps 10,000 and 14,000, the grid is adapted until $\Delta x <$ MFP/2 condition is met for all cells
- There is a large difference of 6 levels of refinement along the cylinder surface

Mach=10, Kn=0.05





- Final, adapted grid has about 22,000 leaf cells and is characterized by about 20 (free stream region) to about 2 (stagnation region) particles per cell.

Delivering Breakthrough Solutions

GPU-Accelerated UFS-DSMC: Tests

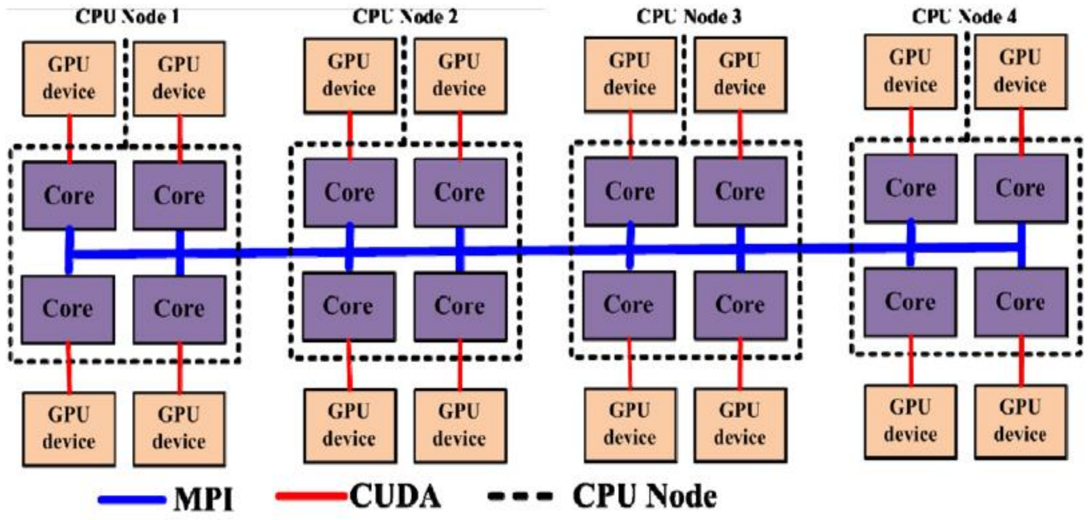
Number of Particles	0.5M	2M
Particle Move Speedup	22	22
Particle Collision Speedup	44	36
Particle Move (CPU), %	42	53
Particle Collision, (CPU), %	35	34
Particle Move (GPU), %	64	70
Particle Collision (GPU), %	27	27
Overall Speedup	29	22

- Overall speedup factor of 22 (real CPU-only time/real GPU-CPU time = 51136 sec/2306 sec) was achieved for the case with 2M particles and 29 for the case with 0.5M particles.
- In particular, the particle movement part speeds up by a factor of 22 and the particle collision part (without the indexing part) – by a factor of 36 for the case with 2M particles.
- Collisional part is accelerated better since no neighbor indexing/retrieving is involved. Collisional kernel speedup is slightly larger in case with 0.5M particles which is most likely due to a lower particle-indexing overhead.

Summary for single GPU device

- GPU accelerated DSMC code (UFS-DSMC-GPU) has been implemented
- Implementation was carried out based on algorithms proposed in the literature for uniform (structured) Cartesian grids and simple embedded bodies
- These algorithms have been extended to adaptive (unstructured) octree Cartesian grids and to solid bodies of arbitrary shape (specified either analytically or from CAD files)
- Corresponding GPU kernels have been developed for each part of code (particle movement, collisions, sampling)
- The code has been tested on a modern GPU device and validated (against CPU-only results) for different problems
- For typical cases of hypersonic flows past blunt bodies in collisional regimes (Knudsen numbers 0.01–0.05), speed up factors of 25–45 were achieved for different parts of the code.

Multiple GPUs: MPI-CUDA Paradigm



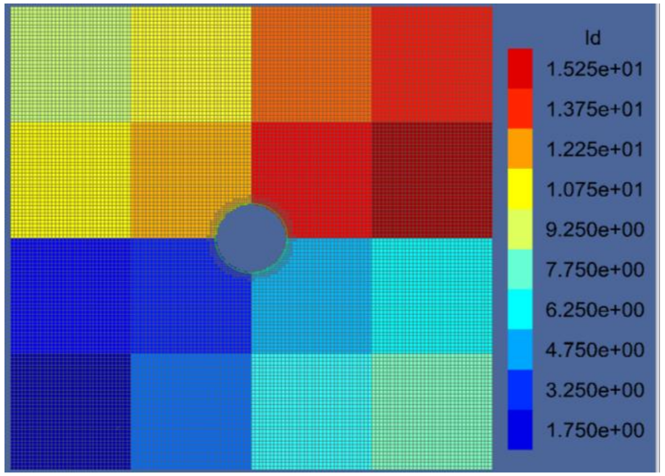
From Su., et al. *J. Comp. Phys.* Vol. 231 (2012) 7932

- Spatial domain decomposition using the Forrest-of-Trees (FoT) method
- MPI protocol to exchange data from memory of all MPI processors and synchronize
- CUDA is used to put the DSMC-related simulation components on GPU and for data transfer between CPU (host) memory and the GPU (device) global memory
- CUDA API function cudaSetDevice() to assign a GPU to each individual MPI process
- For data exchange between global memory of different GPU devices we use the CUDA API function cudaMemcpy() (red line)
- Data is transferred from host-A to host-B (blue line) using the MPI protocol with MPI_Send() and MPI_Recv().
- cudaMemcpy() to transfer data from host-B to device-B.

CFDRC Delivering Breakthrough Solutions

MPI in UFS-DSMC Particle Code

- MPI capability implementation using FoT parallelization algorithms.
- Domain decomposition is based on breaking the computational domain into boxes: each CPU then receives a set of such boxes depending on some partitioning algorithm (e.g., based on a number of cells).
- The FoT is a coarse-grain parallel algorithm since load balancing can be done only in terms of the building (root) boxes.



Initial grid and box ID partioning for a problem of heat transfer with 16 (root) boxes and immersed solid phase

- Each CPU operates on a given number of boxes, which is useful since computational grid is stored only on the host CPU.

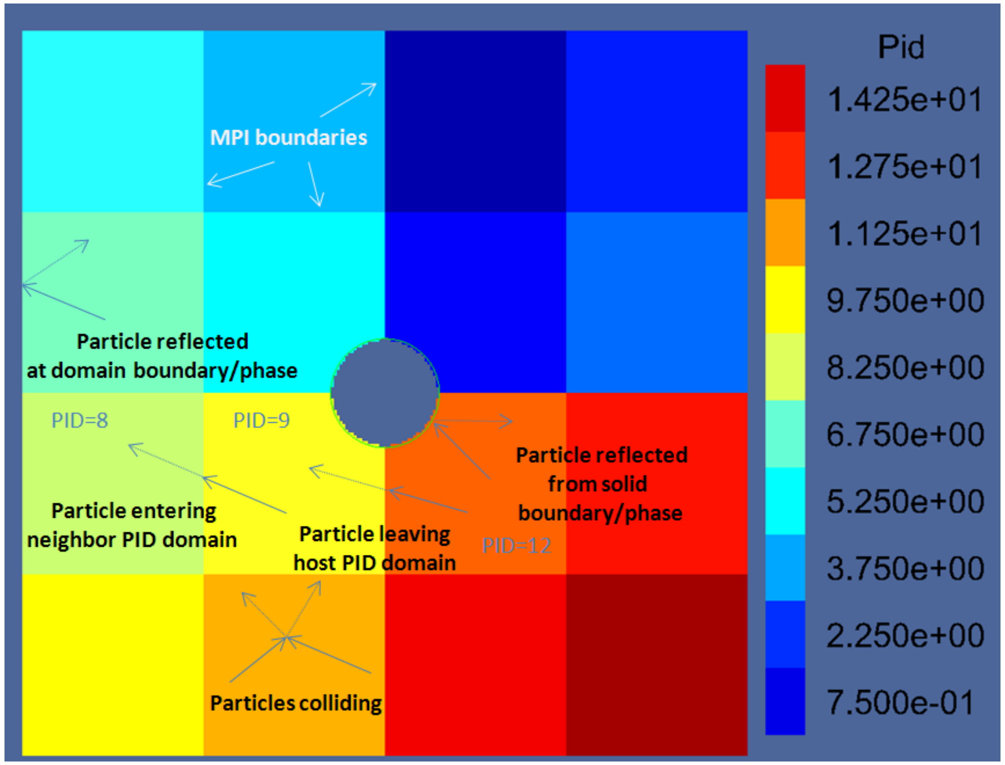
- During grid adaptation and then dynamic load balancing (DLB) these boxes are exchanged between the CPUs according to some balancing algorithm.

- Each box has it own ID number and a set of boxes on each host CPU share the same PID (Processor ID) number.

CFDRC Delivering Breakthrough Solutions

MPI in UFS-DSMC Particle Code

- Particles are initialized in each PID domain. They start to move and interact with each other, with domain boundaries, and with solid surfaces/phases immersed into the domain
- Particles (their position, velocity, remaining move time, etc.) which hit an MPI boundary are stored in a special list
- Knowing direction of box boundary face a particle hits (PID=9), its neighbor PID (e.g., PID=8) is determined
- Neighboring PIDs receive and then add these particles for further processing during the next time step.


MPI boundaries
Particle reflected at domain boundary/phase
PID=8
PID=9
Particle entering neighbor PID domain
Particle leaving host PID domain
Particle reflected from solid boundary/phase
PID=12
Particles colliding
Pid
1.425e+01
1.275e+01
1.125e+01
9.750e+00
8.250e+00
6.750e+00
5.250e+00
3.750e+00
2.250e+00
7.500e-01

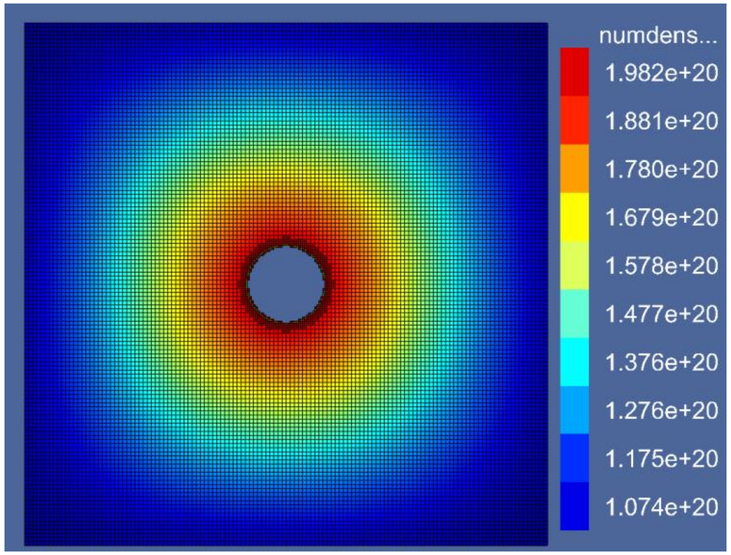
Processor ID partioning for 16 CPU/GPUs, problem of heat transfer. Particle interactions with solid surface/phases, with other particles and with domain and MPI boundaries

- All operations are performed in terms of classes and objects
- MPI exchanges are called via special read and write methods of the objects/classes.
- Exchanges implemented by creating special classes and objects which are inherited from the main GtsObject class (parent class)

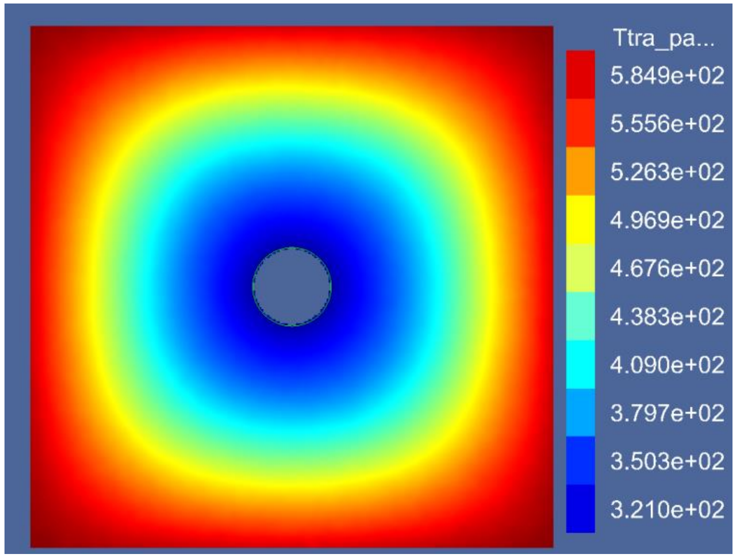
CFDRC Delivering Breakthrough Solutions

MPI in UFS-DSMC Particle Code: Demo

- Heat transfer between a cold cylinder (temperature T_{solid} = 300 K) immersed into a sealed box with hot walls (temperature of the walls T_{wall} = 600 K).
- The (initially uniform) density of gas phase particles corresponds to Knudsen number 0.01.
- Full energy accommodation assumed for particle-wall interactions.
- 30M particles used to achieve good statistics.
- Computational grid with a layer of higher resolution grid around solid surfaces.
- Local cluster node consisting of 8 CPUs (processor Intel(R) Xeon(R) X5675 @ 3.07GHz).



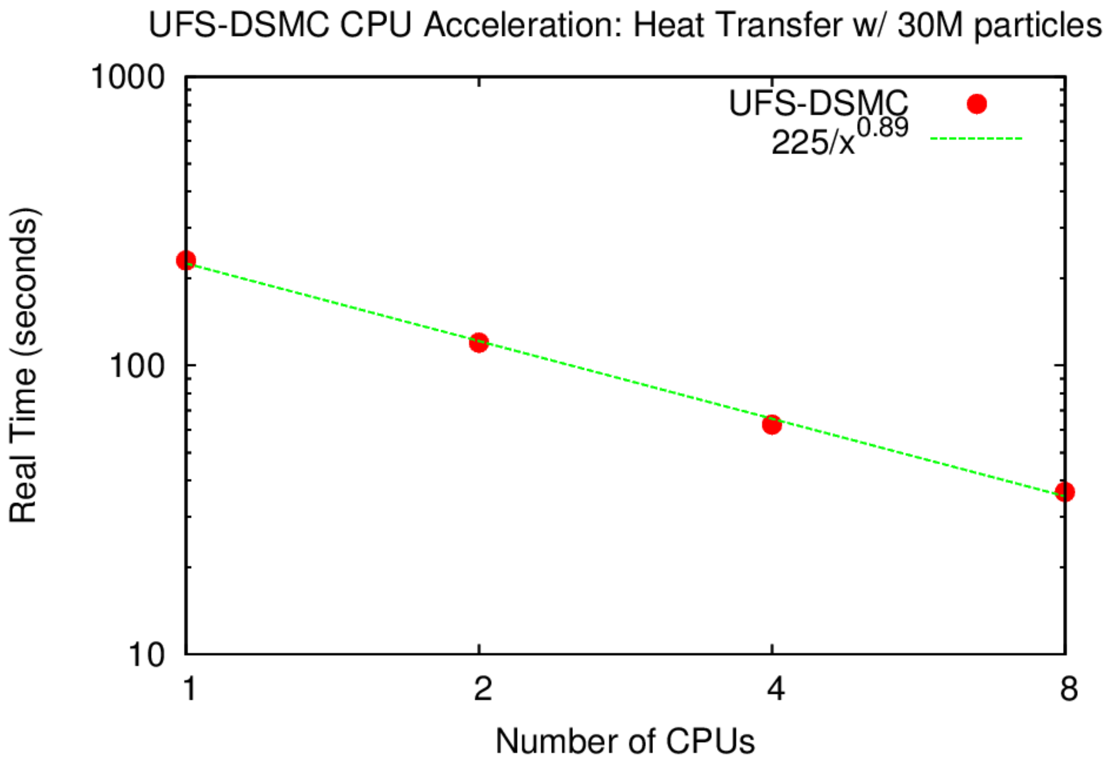
Gas number density



Temperature

MPI in UFS-DSMC Particle Code: Testing

- Real time (required for 200 steps) for different number of CPUs used
- Computational time drops almost linearly (power law factor close to 0.9) with increasing the number of CPUs.
- Almost ideal scaling is achieved for the implemented MPI module.



MPI-CUDA in UFS-DSMC Particle Code

• A GPU kernel is used to calculate the positions of all particles over a time step Δ*t*.

• Each particle's deterministic motion is handled by a CUDA thread, using data held entirely in global memory.

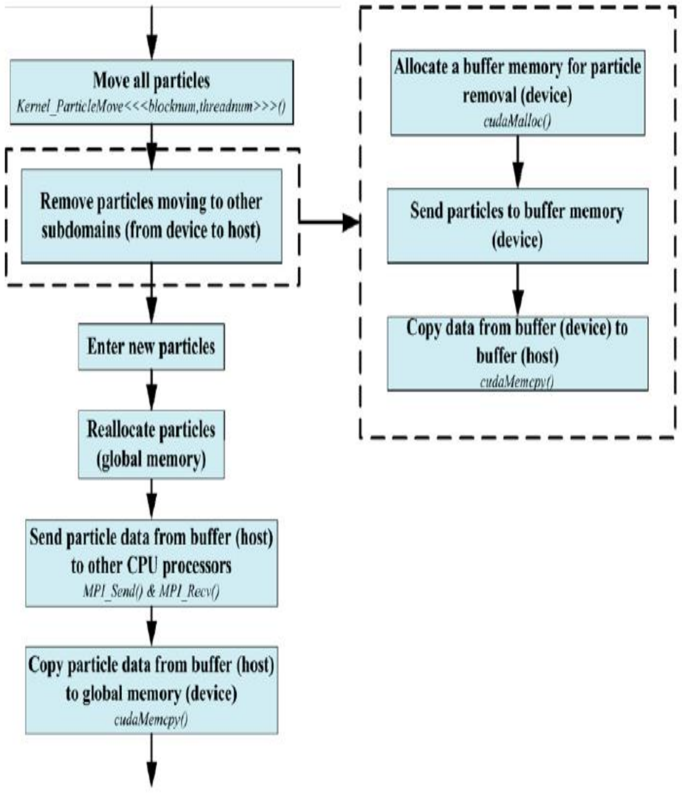
• Particles determined to have left the current GPU's simulation domain are placed into a buffer (in the device, and finally on the host) in preparation for migration to other GPU devices.

• Introduce new particles based on inlet boundary conditions.

• Send and receive from buffers of all MPI processors (host) with the MPI API protocol MPI_Send() and MPI_Recv().

• Reallocate particles from buffers into their newly allocated GPU's global memory.

A flowchart demonstrating the particle movement phase algorithm in the hybrid MPI-CUDA DSMC scheme:



Move all particles
Kernel_ParticleMove<<<blocknum,threadnum>>>()
Remove particles moving to other subdomains (from device to host)
Enter new particles
Reallocate particles (global memory)
Send particle data from buffer (host) to other CPU processors
MPI_Send() & MPI_Recv()
Copy particle data from buffer (host) to global memory (device)
cudaMemcpy()
Allocate a buffer memory for particle removal (device)
cudaMalloc()
Send particles to buffer memory (device)
Copy data from buffer (device) to buffer (host)
cudaMemcpy()

After Su et al. *J. Comp. Phys.* Vol. 231 (2012) 7932.

CFDRC Delivering Breakthrough Solutions

MPI-CUDA in UFS-DSMC Particle Code: Testing

- The MPI-CUDA algorithms was first tested and debugged on a local cluster consisting of 2 Tesla C2075 GPU cards.

- Then, used NASA Pleiades cluster has 64 Westmere nodes each including one NVIDIA Tesla M2090 (512x 1.3GHz cores) GPU.

- Each M2090 GPU device is connected to the CPU node via a PCI Express bus. The nodes are connected via high-speed Infiniband.

The following modules were used to compile and run

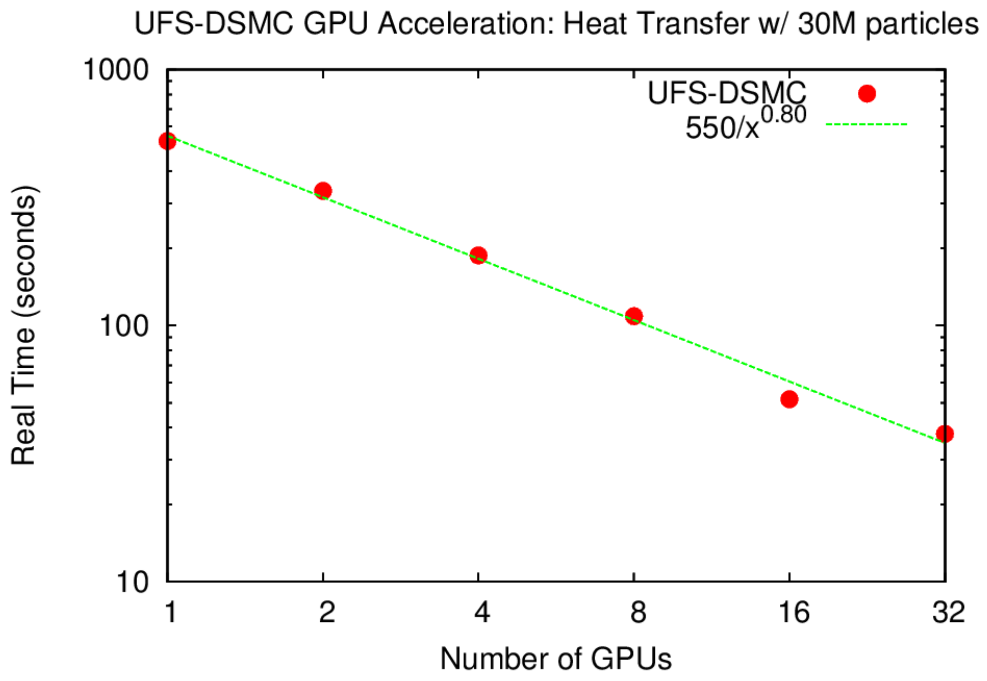
```
module load gcc/4.1.2 mpi-mvapich2/1.4.1/gcc
module load cuda/4.2
```

Runs were carried out in “gpu” interactive queue using 1 CPU/1 GPU per node

```
qsub -I -q gpu -l select=16:ncpus=1:model=wes_gpu -l walltime=0:15:00
```

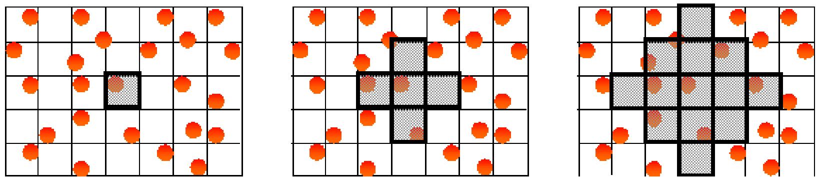
MPI-CUDA in UFS-DSMC Particle Code: Testing

- Same problem of heat transfer used for the CPU-only scaling tests presented above.
- Results obtained on different number of GPUs are identical and they are analyzed for the obtained steady-state solutions (after 40,000 time steps).
- Real time (per 2,500 time steps) as a function of the number of GPUs is shown
- A very good scaling is obtained with the power law scaling factor being 0.8 (factor of 1 means ideal scaling)
- This provides a proof of the high efficiency of the implemented hybrid MPI-CUDA algorithms



Future Work for Lagrangian Particle Module

- Extension to 3D geometries with GPU
- Extension to dynamically adapted grids with GPU (every, say, 100th or 1000th time step, transfer all particles back to CPU, do grid adaptation and cell re-indexing on CPU, put all particles back to GPU, do computations on GPU, and so on)
- Dynamic Load Balancing (based on number of cells and/or number of particles per process) with MPI and GPU. Use MPI box exchange strategies in FOT
- (Two-way) Coupling Eulerian-Lagrangian modules via mutually induced forces.
- Particle collisions for dense flows



(a) Search level 1	(b) Search level 2	(c) Search level 3
Neighborhood size for different search levels		

- **Computation of particle collisions using a search over a local volume.**
- **Particle collisions amongst all the particles contained in the volume defined by these neighboring cells.**
- **A pre-specified search level determines the number of neighboring particles amongst which the collisions are enacted.**

DoE SBIR Phase I Project

Develop a modern adaptive Eulerian-Lagrangian solver for multiphase flows on parallel CPU-GPU clusters with dynamically adaptive Cartesian mesh for high resolution of flow and particle transport.

Specific objectives of Phase I:

- **Develop GPU-accelerated Lagrangian particle transport including:**
  - **GPU processing of particle motion and collisions**
  - **Particle mapping to grid**
  - **Interpolation of local forces from Eulerian fluid and inter-phase momentum transfer**
- **Develop GPU-accelerated Eulerian solver for octree Cartesian mesh**
- **Improve parallelization algorithms with GPU-accelerated construction of Space Filling Curves and Octrees**
- **Develop detailed plan for Phase II implementation, testing and validation as well as marketing and commercialization.**

Acknowledgements

Financial support provided by the DoE SBIR Phase I Project DE-SC0010148 “GPU-Accelerated Multiphase Eulerian-Lagrangian Solver with Adaptive Mesh Refinement”