International Supercomputing Conference 2009

Implementation of a Lattice-Boltzmann-Method for Numerical Fluid Mechanics Using the nVidia CUDA Technology

E. Riegel, T. Indinger, N.A. Adams

Technische Universität München
Institute of Aerodynamics

23.06.2009

Outline

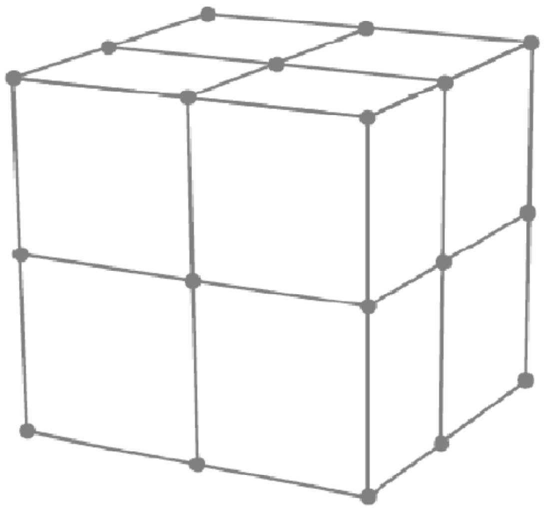
- The Lattice Boltzmann Method (LBM)
- SunlightLB CUDA port
  - Activities
  - Performance evaluation
- LBultra, LBM reimplementation with focus on CUDA
  - Activities
  - Performance evaluation
  - Physical Validation

Outline

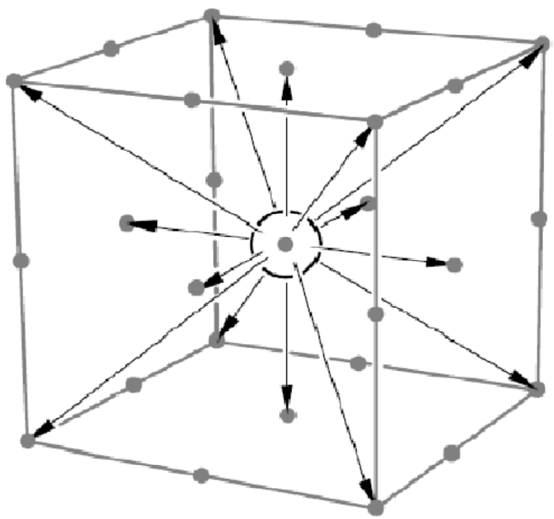
- LBultra, outlook on ongoing work
  - Adaptive mesh refinement
  - Expected Performance
- Conclusions

Lattice-Boltzmann-Method - Discretisation

- Computational fluid mechanics algorithm for incompressible flows (M < 0.3)
- Spatial discretization by partitioning computational space into cubes
- Discrete molecular flows (D3Q15, D3Q19, ..)
- Flow state representation by a discrete distribution function mapping of density onto the discrete molecular flow vectors



D3Q15



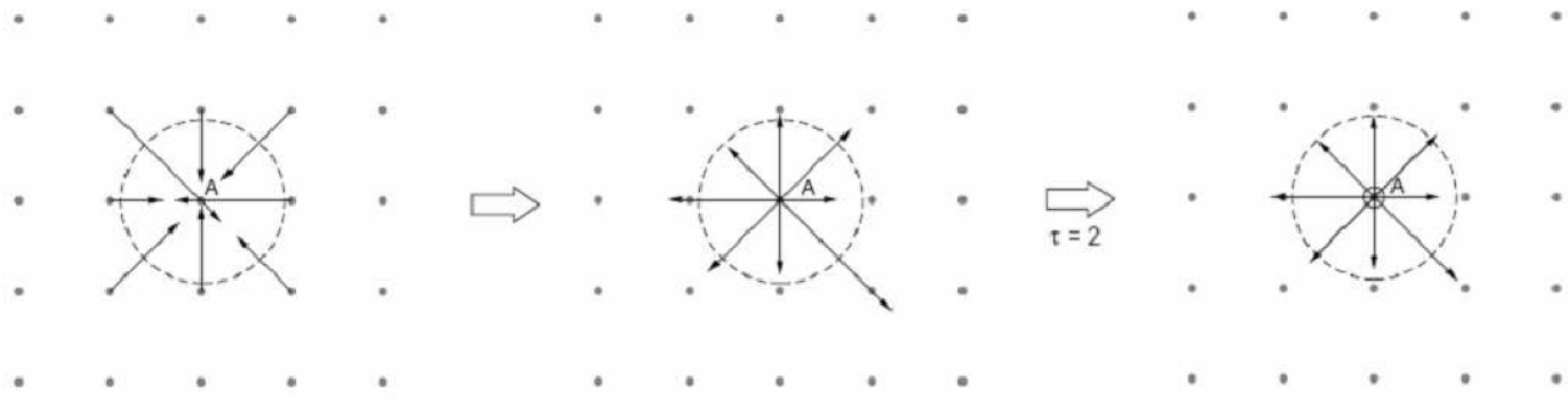
TUM aer

Lattice-Boltzmann-Method - Process

- **Propagation step** transfers distribution function density entries along their molecular velocities to adjacent nodes

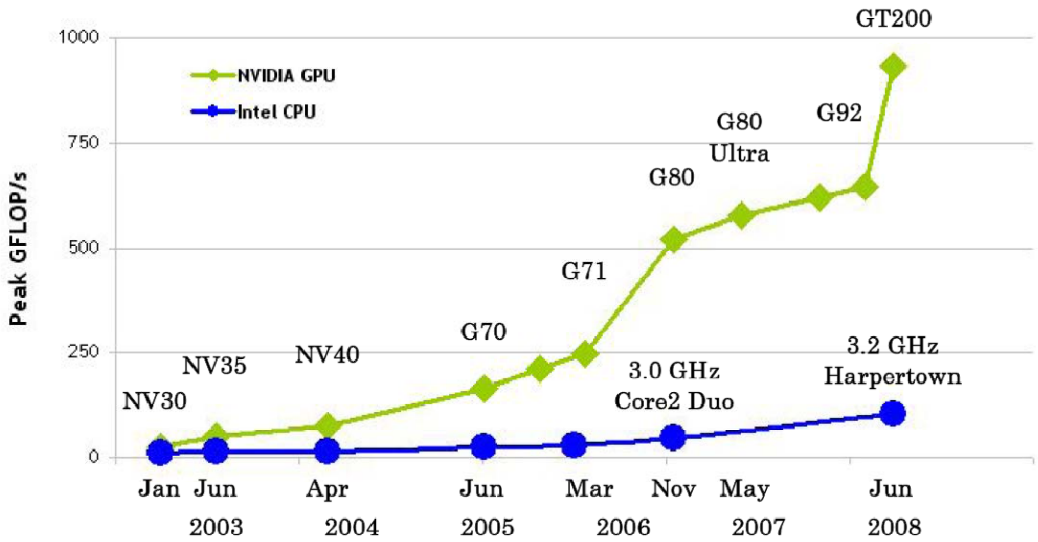
- **Collision step** calculates the collision of distribution function density entries arriving from adjacent nodes in the middle of current node

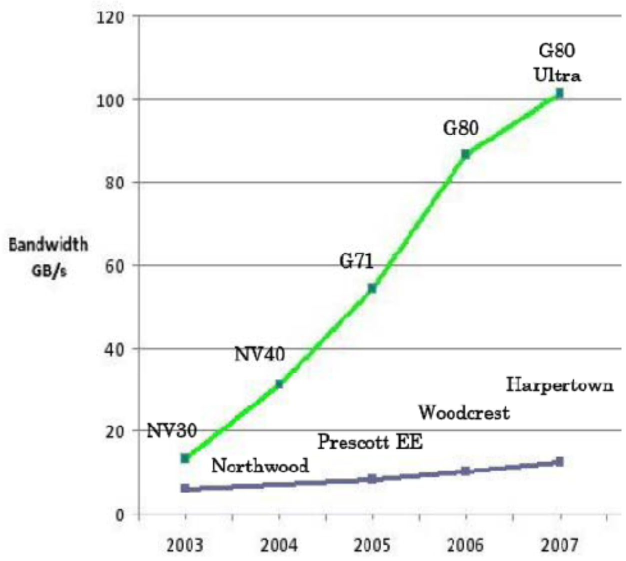
- **Propagation step** of the next time step



LBM – Why CUDA implementation?

- LBM is a perfectly parallel algorithm, each node can be processed independently
  - ➢ Very suitable for CUDA's highly parallel execution model
- LBM allows efficient domain decomposition because of weak sub domain coupling
  - ➢ Low level of simulation-data exchange between sub domain processors
  - ➢ Very suitable for distribution among multiple CUDA devices (multi GPU)





SunlightLB CUDA port - Activities

- SunlightLB
  - open-source LBM D3Q15
  - traditional CPU implementation in C programming language
  - http://sunlightlb.sourceforge.net/

- Porting procedure
  - focusing on core simulation steps
  - adaptation of SunlightLB's CPU algorithms to CUDA's highly parallel execution model
  - writing of glue code for data exchange between HOST and GPU device

- Expected performance increase
  - GPU: GeForce 8800GTS (~450 GigaFLOPs)
  - CPU: Core2Duo 3.7 GHz (OC) (~30 GigaFLOPs)
  - expected speed-up: ~**15x**

SunlightLB CUDA port – Performance analysis

- Benchmark run
  - Domain size: 64x64x64 Voxels
  - Obstacle: Sphere in the middle with radius 16 voxels
  - CPU performance: 9.0 MVPS (millions of voxels per second)
  - GPU performance: 13.4 MVPS
  - Resulting speed-up: **1.5x**

- Expected speed up missed by a factor of 10
  - porting made LBM algorithm kernel highly parallel ⇒ good
  - porting did not take into account CUDA memory access patterns ⇒ **bad**
  - bad GPU memory access patterns can cause a performance dropdown of up to a factor of 32

➢ simple porting of algorithms is not sufficient for high GPU performance, deeper optimization on GPU architecture is necessary

LBultra - Activities

- Completely new implementation of a LBM multi-architecture simulation software
- written in C++ to benefit from Object-Oriented (OO) technology
- consists of a framework and pluggable LBM kernels
- any LBM kernel can be implemented and optimized independently
  - D3Q15 fixed refinement CUDA kernel
  - D3Q15 fixed refinement CPU multi core kernel
- supports domain decomposition by gluing kernels together
- static objects can be placed into computational space
- kernels are enclosed by a shell of boundary data specification structures

LBultra - Activities

- Strategies to a better new CUDA LBM kernel:
  - more attention on memory access patterns
  - algorithmic reduction of data transfers by joined propagation and collision phase
  - using shared GPU memory for explicit data caching

- New CPU LBM kernel:
  - has been derived from the GPU kernel by “porting back”
  - offers high parallelism as well -> can utilize multi core CPUs
  - also profits from algorithmic improvements in CUDA kernel

- Expected performance increase:
  - GPU: nVIDIA Tesla C1060 (~933 GigaFLOPs)
  - CPU: AMD Phenom X4 2.6Ghz (~41 GigaFLOPs)
  - Expected Speed-up: **~23x**

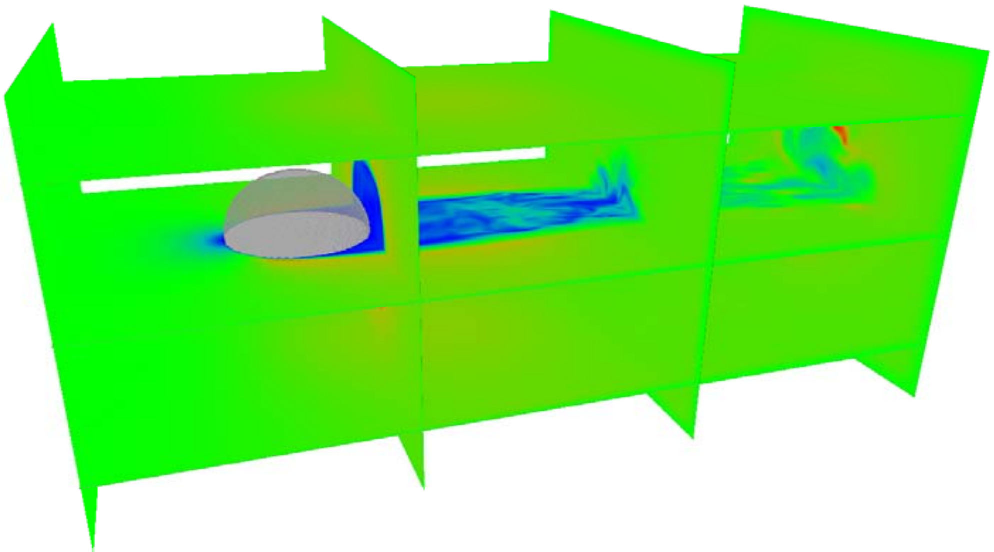
LBultra - performance

- Benchmark run
  - Domain size: 384x384x384 voxels
  - Obstacle: Sphere in the middle with radius 64 voxels
  - CPU performance: 8.42 MVPS
  - GPU performance: 78.0 MVPS
  - 3 x GPU performance: 191 MVPS
  - Resulted speed-up for one GPU: **9,3x**

- Performance analysis:
  - performance much better, but still not optimal
  - reason: bad memory access pattern in “collection” of distribution function entries for propagation step
  - further optimization of LBM algorithms for CUDA architecture should yield in higher performance
  - domain linkage GPU algorithm not optimal yet, without domain linkage GPU performance is around 110 MVPS

LBultra - validation

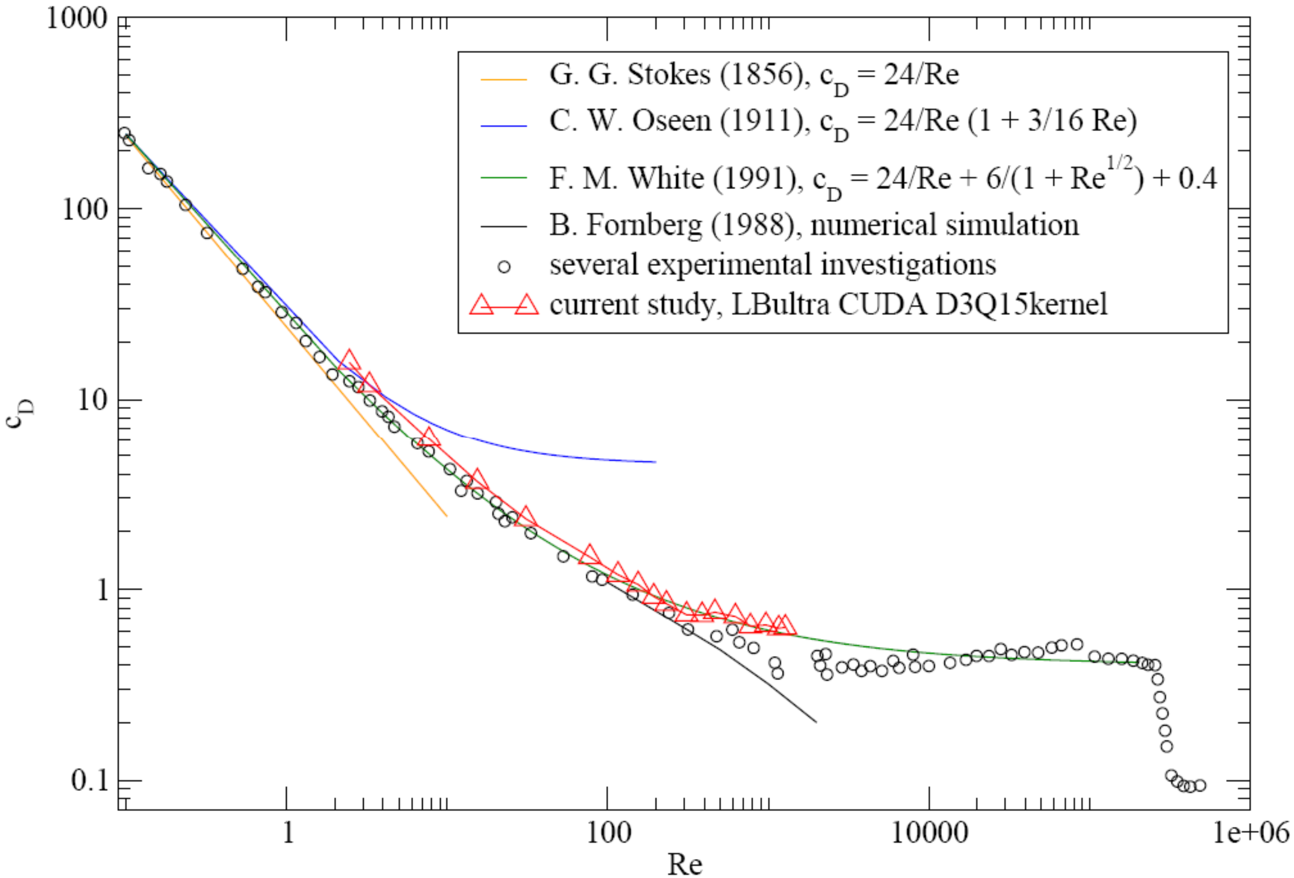
- Validation scenario:
  - common test case of a flow around a sphere
  - testing Re-Range from Re=2.46 to Re=1280
  - inflow: homogenous velocity, equal to average velocity in adjacent plane
  - other: zero normal velocity gradient
  - flow acceleration by a volumetric acceleration
  - computation of approx. 10000 time steps to enable convergence
  - c_D measurement and averaging for 1000 further time steps
  - graphical comparison with data from other numerical simulations and experiments



LBultra - validation

- Results:

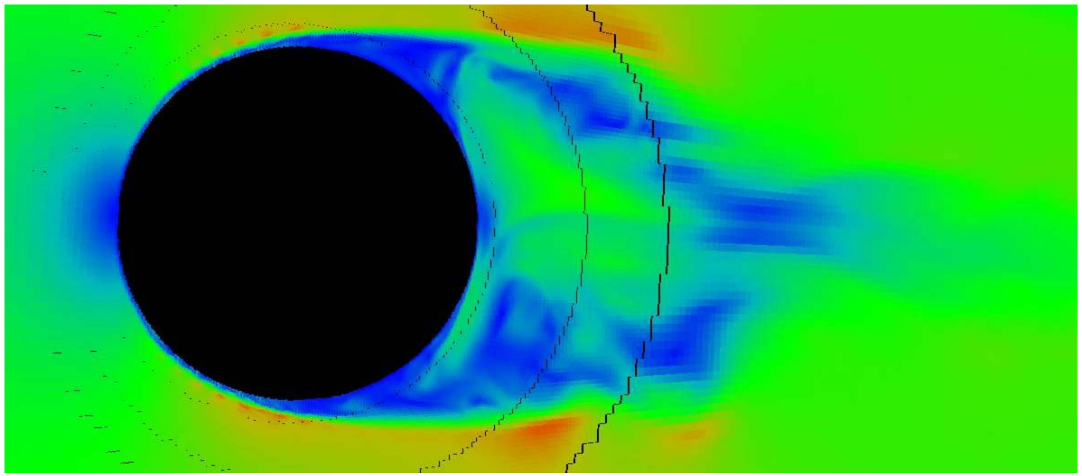
Re	c_D
1280	0.629
1160	0.623
966	0.648
774	0.639
624	0.720
465	0.759
387	0.732
310	0.736
232	0.841
194	0.916
155	1.06
116	1.19
77.4	1.47
31.0	2.33
15.4	3.66
7.72	6.14
3.28	11.9
2.46	15.6

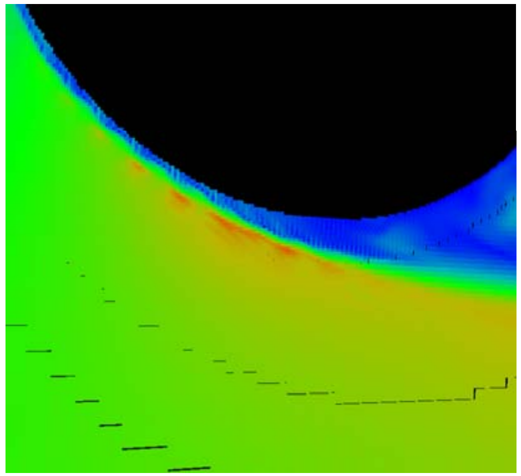


-> Excellent agreement with reference data.

LBultra – outlook on ongoing work

- Enhancement of adaptive mesh refinement capability to CPU and CUDA kernels
  - Node resolution is location dependent
  - resolution is selected by the fluid or geometry requirement at any location
  - improves computation time and data efficiency by not wasting resources for locations were nothing is happening
  - Data and computation time amount depends by $O(n^3)$ from the node resolution!!!





LBultra – outlook on ongoing work

- CPU kernel already capable of single threaded computations with local mesh refinement
  - expected multi core CPU (~41 GigaFLOPs) performance : ~10 MVPS

- CUDA kernel is currently under development
  - not usable yet, but first computation stages already working
  - new block-based LBM algorithm is expected to work better on GPU architectures
  - expected CUDA single GPU (~933 GigaFLOPs) performance: ~400 MVPS

Conclusions

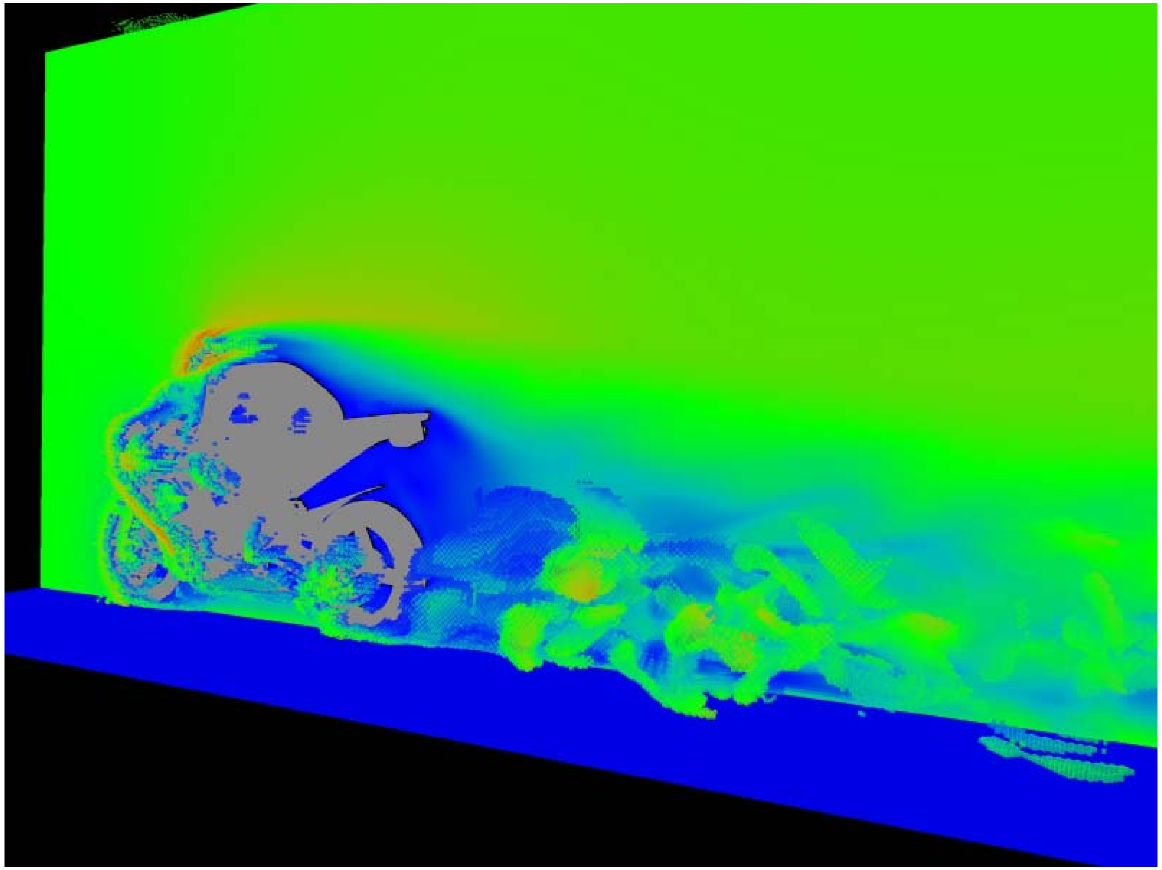
- Lattice-Boltzmann-Method is well suited for implementation with CUDA
  - high computational performance achievable
  - valid simulation results
- C-like CUDA language encourages porting of existing software to CUDA
- efficient CUDA software rather requires a new implementation with special optimizations than a simple port
- CUDA hardware has some important advantages in compare to traditional CPU architecture
  - 8x to 20x faster (depends on algorithm and optimization effort)
  - 6x more favorable price
  - consumes 7x to 19x less space (nVIDIA Tesla S1070 4xGPU, 1HE)
  - consumes 4x less electrical power

➢ Additional effort leads to considerable gain

Acknowledgements

The authors are especially grateful to **nVIDIA** who supported this work by providing GTX 280 for initial testing and implementation and Tesla C1060 hardware for the final validation runs.

Thank you very much for your attention!



Lattice-Boltzmann-Method – boundary data setup

- Static obstacle geometry is considered by “reflection” of distribution function density entries along molecular velocities passing an obstacles outline

- Distribution functions of in- and outflows is calculated using given macroscopic velocity, density and stresses