Fluid dynamics simulation with Lattice Boltzmann Models using CUDA enabled GPGPUs

Adrian HORGA

Supervisors:

Dr. Fiz. Victor SOFONEA

Conf. Dr. Ing. Marius MINEA

Lattice Boltzmann Models (LBM)

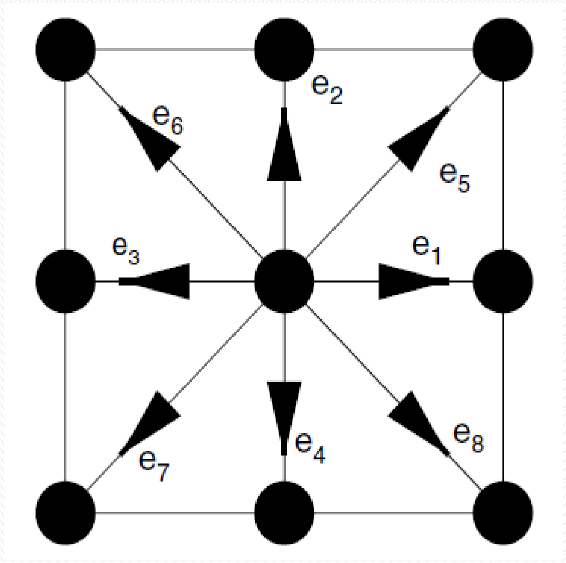
- Method for representing fluid dynamics

- Reduces complexity
  - limited number of velocities
  - discrete time steps
  - works with probabilities – e.g. probability density function

- Fluid is lattice based (not particle based)
  - a clear lattice layout of the fluid is specified
  - the fluid characteristics (velocity, density, temperature etc.) are simulated in each lattice node

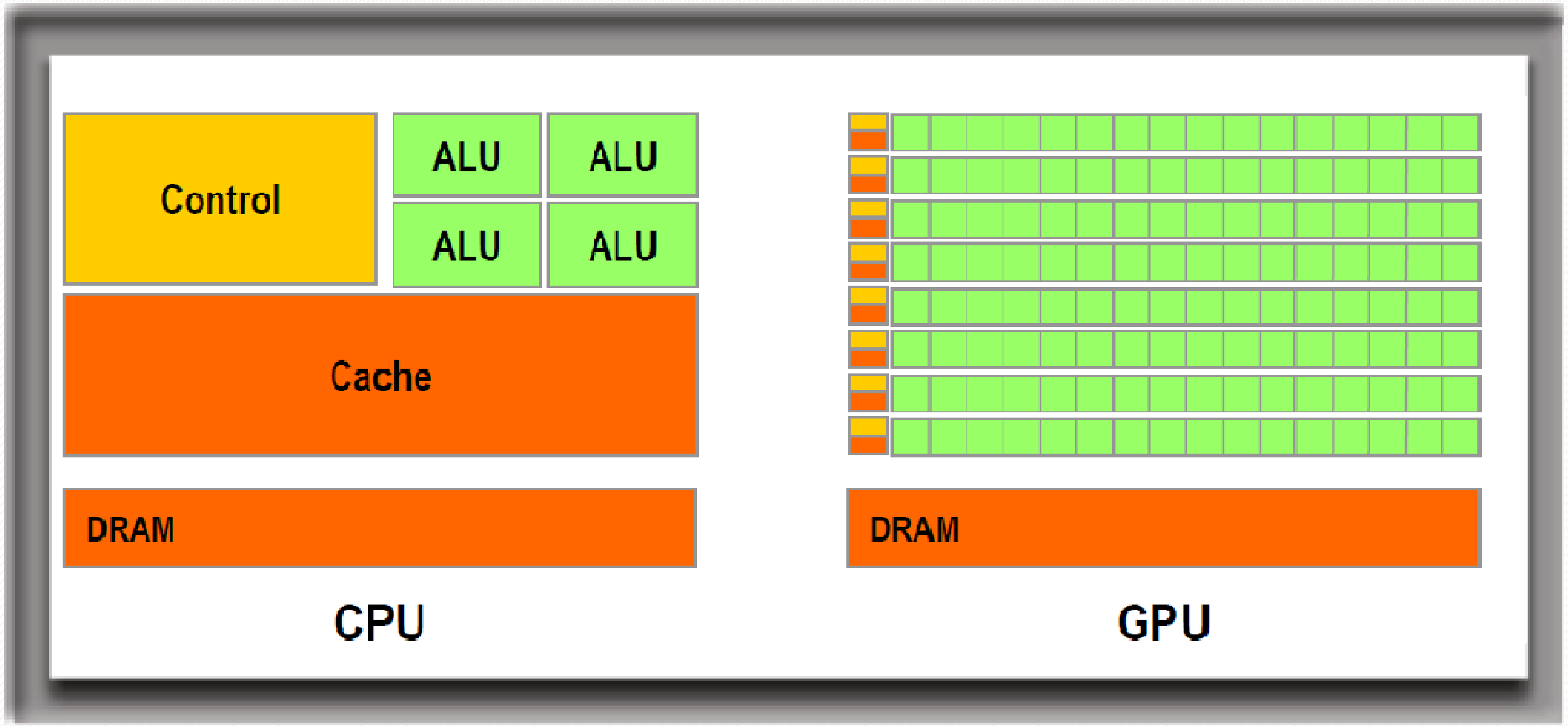
DnQm model

- n – dimensions
- m – number of the velocity vectors



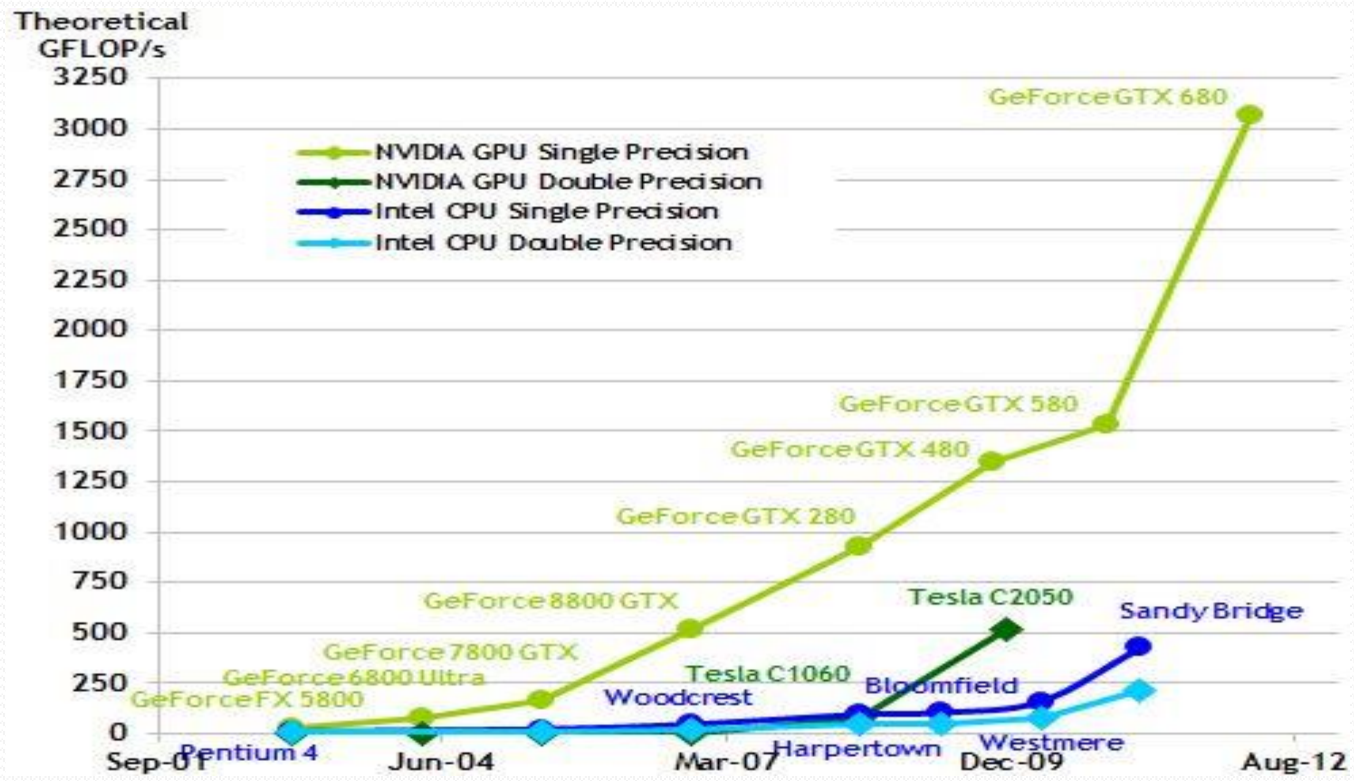
Possible orientations of particle velocities in the D2Q9 model (S. Succi, 2001)

CPU vs GPU



NVIDIA CUDA C Programming Guide, v. 5.0, 2012

CPU vs GPU



NVIDIA CUDA C Programming Guide, v. 5.0, 2012

CUDA toolkit

- Special compiler : nvcc
- C based functions for operations like:
  - memory copy :
    - cudaMemcpy()
  - kernel invocation :
    - myKernel<<<blocks, threads>>>(args)
- __global__ identifier
  - visible on host(CPU)
  - runs on device(GPU) – kernels are defined with this identifier
- __device__ identifier
  - visible only on device

CUDA LBM – previous work

- Streaming collision – simpler equations compared to finite difference
- No force term
- Single phase fluids
- Bounce back boundary conditions on solid walls

- J. Tolke , 2009 - D2Q9 - Single precision
- J. E. McClure et al., 2010 - D3Q19 - Single precision
- L. Biferale et al., 2011 - D2Q37 - Double precision

Our model –finite difference LBM

- Used for multiphase fluid (liquid vapor)
- Has a force term to be computed in every lattice node
  - more complex
  - more accurate
- Use a periodic boundary condition
  - edge nodes use data from the opposite side
- Compute data from 2 lattice distance
  - use a 17-node stencil to compute different elements like : pressure, force, etc. (maximum distance of 2 lattice space)

CUDA implementation for D2Q9 LBM

- Starting formulas from PETSc implementation of V. Sofonea, 2005 -> transform the code for CUDA
- Each iteration – 2 steps
  - compute all necessary data for each node
  - compute the probability distribution functions (related to each velocity) for the next iteration based on the data from the first step
- for i = 0, iterations

```
        {
        periodic_boundary<<<>>>()
        lb_time_step<<<>>>()
        }
```

CUDA implementation for D2Q9 LBM – Improvement possibilities

- Possible speed-ups – CUDA Best Practices
  1. Use single precision
     - twice as fast as double precision
  2. Reduce the global memory usage
     - higher bandwidth
  3. Reduce register usage per kernel
     - increased occupancy
  4. Memory coalescing – adjacent kernels call adjacent memory locations
     - higher bandwidth

CUDA implementation for D2Q9 LBM – Improvement possibilities

- Possible speed-ups – CUDA Best Practices

5. Use shared memory
   - 10x faster than global memory

6. Increase occupancy
   - more threads than run in parallel

7. Reduce branching (no divergent if, do, while statements for threads in a warp -> 32 threads)
   - higher operation throughput

CUDA implementation for D2Q9 LBM - Improvements

1. Use single precision
   - we need increased accuracy -> use double precision

2. Reduce the global memory usage
   - we use more registers to store the global memory variables

3. Reduce register usage per kernel
   - recalculate certain elements like position index

4. Memory coalescing
   - we linearize the two dimensional table and use only one index
   - because we use the 17-node stencil we cannot have coalescent memory on all the memory calls

CUDA implementation for D2Q9 LBM - Improvements

5. Use shared memory
   - limited resource - 64kb for each streaming multiprocessor
   - too many nodes to compute (more than 9 * 256 * 256) – not feasible

6. Increase occupancy
   - reuse threads to compute another node after finishing
   - we do not need as many threads as nodes (e.g. 1024*1024)

CUDA implementation for D2Q9 LBM - Improvements

7. Reduce branching (no divergent if, do, while statements for threads in a warp -> 32 threads)
   - by tiling the auxiliary(old values) matrix with "ghost" nodes – 2*2 rows and 2*2 columns
     - we do not have to check the position of each node (if it is on the edge or not) – useful for later periodic boundary conditions
     - improves memory coalescing

CUDA implementation for D2Q9 LBM

- Improvement of the algorithm
  - starting from the original PETSc version
  - modify for CUDA
  - follow steps 1 - 7

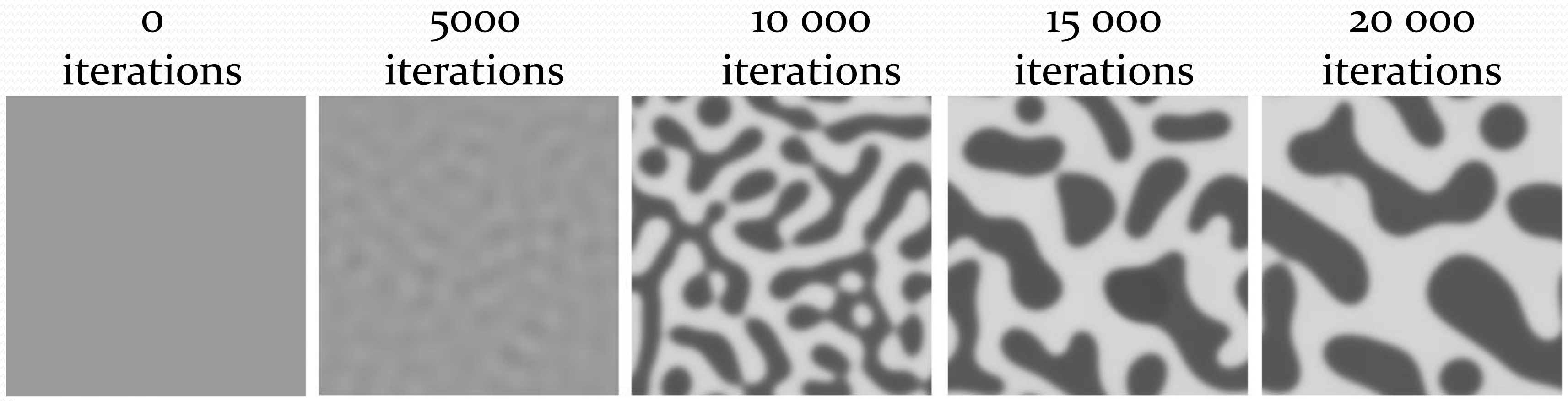
- Steps that required a lot of work
  - 2. Reduce the global memory usage
  - 3. Reduce register usage per kernel
  - 6. Increase occupancy

Two versions

- V1 - improvements specified before
- V2
  - further reuse of registers -> obfuscates the code
  - recalculating indexes, certain sums

- Test on 2 devices
  - GTX460 – laptop GPU (168 GFLOPS double precision)
  - Tesla M2090 – powerful scientific GPU (665 GFLOPS double precision)

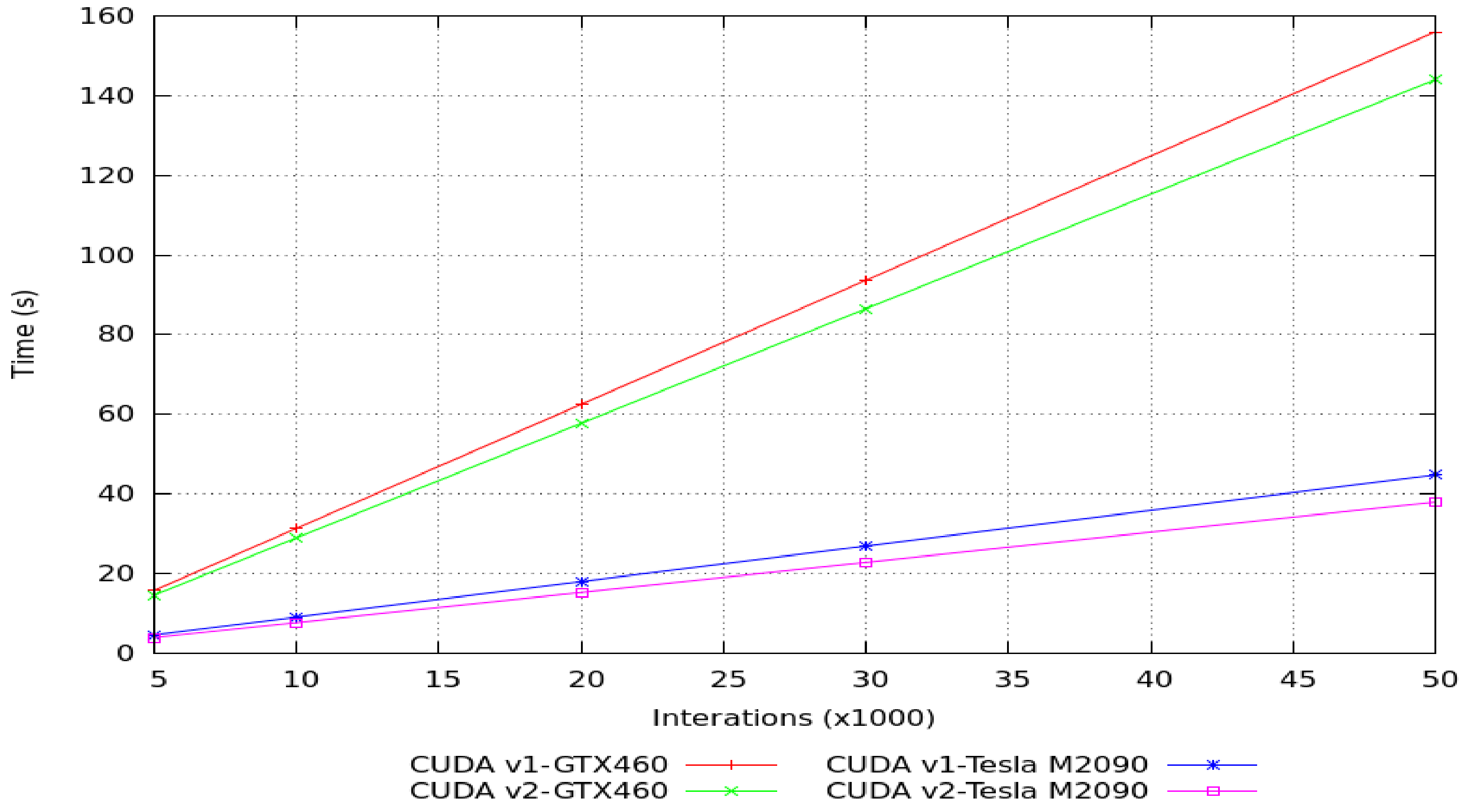
Results – phase separation



Liquid (black) - vapor (white) phase separation within a 256x256 matrix

Results – 256 * 256

Running time preserves 4:1 performance ratio of graphics cards



Compare to the PETSc implementation

- Multi-CPU implementation
- Runs on IBM BlueGene/P supercomputer
- Double precision

http://hpc.uvt.ro/wiki/BlueGene

Compare: 16 cores <-> CUDA 52 times faster

256x256	Time (s)				
Iteration	PETSC – 16 cores	CUDA v1 – GTX460	CUDA v2 – GTX460	CUDA v1 – Tesla M2090	CUDA v2 – Tesla M2090
5000	199.092	15.653	14.461	4.47	3.789
10000	397.398	31.249	28.869	8.944	7.574
20000	793.755	62.419	57.659	17.878	15.145
30000	1189.43	93.592	86.453	26.81	22.718
50000	1982.489	156.039	144.06	44.722	37.861

Compare: 8x more cores (128) <-> CUDA 13 times faster (only 4x performance increase on PETSc)

512x512	Time (s)				
Iteration	PETSC – 128 cores	CUDA v1 – GTX460	CUDA v2 – GTX460	CUDA v1 – Tesla M2090	CUDA v2 – Tesla M2090
5000	203.573	68.065	58.323	17.795	15.664
10000	417.739	136.2	116.662	35.579	31.319
20000	828.849	272.329	233.188	71.134	62.618
30000	1249.349	408.333	349.978	106.701	93.911
50000	2062.281	680.45	583.007	177.849	156.516

Compare -> CUDA as fast as 2048 cores

- Possibly …
  - we can't use all the resources of the supercomputer
  - but we can fully use the resources of a single GPU

Compare to published results

- MLUPs – million lattice updates per second

- Our solution - D2Q9 - DP - 84 MLUPS
  - Tesla M2090(665 GFLOPS DP)
    - 0.126 MLUPS/GFLOPS
- L. Biferale et al., 2011 - D2Q37 - DP - 20 MLUPS
  - Tesla C2050(515 GFLOPS DP)
    - 0.038 MLUPS/GFLOPS
- J. Tolke , 2009 - D2Q9 - SP - 568 MLUPS
  - GeForce 8800 Ultra(410 GFLOPS SP)
    - 1.385 MLUPS/GFLOPS
- J. E. McClure et al., 2010 - D3Q19 - SP - 250 MLUPS
  - Quadro FX 5600(345 GFLOPS SP)
    - 0.724 MLUPS/GFLOPS

Conclusions - CUDA

- When using periodic boundary conditions –> “ghost” nodes improve the memory coalescing
- Hard to optimize
  - finite difference LBM equations severely impact performance (MLUPS) -> new memory access patterns should be studied
  - different CUDA versions imply different approaches because of the resources and the API
    - portable code -> not optimized
    - optimized code -> portable but may need different optimizations on newer versions or different machines
  - occupancy is the most important element -> done correctly can hide poor memory access patterns

Questions?

Acknowledgements

- This work was supported by a grant of the National Authority for Scientific Research

- CNCS-UEFISCDI project number PN-II-ID-PCE-2011-3-0516

- Host Institution:
  - Romanian Academy -- Timisoara Branch
    - Center for Fundamental and Advanced Technical Research

- Project director: Dr. fiz. Victor Sofonea

References

- S. Succi, "*The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*". Oxford University Press. ISBN 0-19-850398-9, 2001
- V. Sofonea, R. F. Sekerka, "Boundary conditions for the upwind finite difference Lattice Boltzmann model: Evidence of slip velocity in micro-channel flow", Journal of Computational Physics 207, 639–659, 2005
- NVIDIA. NVIDIA CUDA C Programming Guide, v. 5.0, 2012
- Jonas Tolke. "Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by NVIDIA". Comput. Vis. Sci., 13(1):29–39, 2009.
- J. E. McClure, J. F. Prins, C. T. Miller, "*Comparison of CPU and GPU 9 Implementations of the Lattice Boltzmann Method*", XVIII International Conference on Water Resources, CMWR 2010, Barcelona
- L. Biferale, F. Mantovani, M. Pivanti, F. Pozzati, M. Sbragaglia, A. Scagliarini, S. F. Schifanoc, F. Toschi, R. Tripiccione, "*An optimized D2Q37 Lattice Boltzmann code on GP-GPUs*", *http://dx.doi.org/10.1016/j.compfluid.2012.06.003*