SIAM PP’2014

High Scalability of Lattice Boltzmann Simulations with Turbulence Models using Heterogeneous Clusters

C. Riesinger, A. Bakhtiari, M. Schreiber
Technische Universität München
February 20, 2014
Topics

Motivation

The Lattice Boltzmann Method

Optimization
  Single GPU optimization: A-B vs. A-A Pattern
  Multi GPU optimization: Communication Schemes

Results
  Runtime
  Lattice Updates
  Parallel Efficiency

Conclusion & Outlook
Topics

Motivation

The Lattice Boltzmann Method

Optimization
- Single GPU optimization: A-B vs. A-A Pattern
- Multi GPU optimization: Communication Schemes

Results
- Runtime
- Lattice Updates
- Parallel Efficiency

Conclusion & Outlook
Motivation

Main Goal
Development of a highly scalable code for heterogeneous clusters to investigate new ideas.

Minor Goal
Extend the software step by step with additional features (never loosing the main goal out of sight).
Motivation

Main Goal
Development of a highly scalable code for heterogeneous clusters to investigate new ideas.

Minor Goal
Extend the software step by step with additional features (never loosing the main goal out of sight).

Properties of our Software

- application: Computation Fluid Dynamics
- algorithm: Lattice Boltzmann Method
- heterogenous system: CPUs + GPUs
- development platform: OpenCL

⇒ Yet another Lattice Boltzmann Code
Features implemented so far:

- Implementation of LBM
- Tailored software design for heterogeneous clusters
- Overlapping communication with computation
- Support for free surfaces
- Considering turbulent flows
- Validation of the simulation results
Features

Features implemented so far:

- Implementation of LBM
- Tailored software design for heterogeneous clusters
- Overlapping communication with computation
- Support for free surfaces
- Considering turbulent flows
- Validation of the simulation results

Features to be implemented:

- Support for (complex) geometries
- Investigation of additional turbulence models
- Exploiting CPU power
- ...
Features

Features implemented so far:
- Implementation of LBM
- Tailored software design for heterogeneous clusters
- Overlapping communication with computation
- Support for free surfaces
- Considering turbulent flows
- Validation of the simulation results

Features to be implemented:
- Support for (complex) geometries
- Investigation of additional turbulence models
- Exploiting CPU power
- ...
The Lattice Boltzmann Method

Motivation

The Lattice Boltzmann Method

Optimization
  Single GPU optimization: A-B vs. A-A Pattern
  Multi GPU optimization: Communication Schemes

Results
  Runtime
  Lattice Updates
  Parallel Efficiency

Conclusion & Outlook
Boltzmann Equation

\[ \frac{\partial f}{\partial t} + \vec{v} \cdot \nabla f = \Delta (f - f^{eq}) \]

Collision operator

\[ f(\vec{x}, \vec{v}, t): \text{ Probability density function:} \]
\[ \text{Probability for finding particles at } \vec{x} \text{ and time } t \text{ with velocity } \vec{v} \]

\[ f^{eq}: \text{ Equilibrium distribution} \]
The Lattice Boltzmann Method 1/3

Boltzmann Equation

\[
\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla f = \Delta \left( f - f^{eq} \right)
\]

Collision operator

\( f(\vec{x}, \vec{v}, t) \): Probability density function:
- Probability for finding particles at \( \vec{x} \) and time \( t \) with velocity \( \vec{v} \)

\( f^{eq} \): Equilibrium distribution

Lattice Boltzmann Method (LBM)

LBM simulates fluid flow by...
- tracking virtual particles (more specifically, their distribution)
- colliding and streaming them (collision & streaming phase)
- only along a limited number of directions (DdQq discretization schemes)
The Lattice Boltzmann Method 2/3

Discretization Schemes

\[ d: \text{ number of dimensions} \]
\[ q: \text{ number of density distributions/lattice vectors} \]

Collision & Streaming

During collision phase, the collision operator is approximated (BGK model):

\[
f_i(\vec{x}, t + \delta t) = f_i(\vec{x}, t) + \frac{1}{\tau}(f_i^{eq} - f_i)
\]

\[
f_i^{eq}(\vec{u}) = \omega_i \rho \left( 1 + 3(\vec{e}_i \cdot \vec{u}) + \frac{9}{2}(\vec{e}_i \cdot \vec{u})^2 - \frac{3}{2} \vec{u}^2 \right)
\]

with \( \omega_i = \frac{1}{18} \) for \( i \in 0, \ldots, 3, 16, 17 \), \( \frac{1}{36} \) for \( i \in 4, \ldots, 15 \), and \( \frac{1}{3} \) for \( i = 18 \).

Movement of the virtual particles is simulated during streaming phase:

\[
f_i(\vec{x} + \vec{e}_i, t + \delta t) = f_i(\vec{x}, t + \delta t)
\]
The Lattice Boltzmann Method 3/3

Densities & Velocities

Only density distributions along the lattice vectors are stored. Densities & velocities are NOT explicitly stored per cell, but can be calculated:

\[ \rho = \sum_{i=0}^{18} f_i \]
\[ \rho \vec{v} = \sum_{i=0}^{18} (f_i \vec{e}_i) \]

LBM for High Performance Computing

- Data access only for discretized density distributions
- Communication only with neighboring cells
- Calculations only use “simple” instructions
Topics

Motivation

The Lattice Boltzmann Method

Optimization

Single GPU optimization: A-B vs. A-A Pattern
Multi GPU optimization: Communication Schemes

Results

Runtime
Lattice Updates
Parallel Efficiency

Conclusion & Outlook
Single GPU optimization: The A-B Pattern

Collision

- Density distributions of own cell are read
- Collision operator is executed
- Updated density distributions are written
Single GPU optimization: The A-B Pattern

**Collision**
- Density distributions of own cell are read
- Collision operator is executed
- Updated density distributions are written

**Propagation**
- Updated density distributions are copied to the corresponding neighboring cells
Single GPU optimization: The A-B Pattern

Collision

- Density distributions of own cell are read
- Collision operator is executed
- Updated density distributions are written

Propagation

- Updated density distributions are copied to the corresponding neighboring cells

To avoid race conditions, two dedicated arrays for the density distributions are required! They are used in a ping pong like manner.
Single GPU optimization: The A-A Pattern

Only one memory layout is used for all iterations. This saves memory by a factor of two \(\Rightarrow\) twice as many cells per GPU

\(\alpha\) Step (odd iterations)

- Performed during odd iterations
- Only one collision is performed
- Memory belongs to own cell

\(\beta\) Step (even iterations)

- Two streamings and one collision
- Memory belongs to neighbors
Single GPU optimization: The A-A Pattern

Only one memory layout is used for all iterations. This saves memory by a factor of two ⇒ twice as many cells per GPU

α Step (odd iterations)
- Performed during odd iterations
- Only one collision is performed
- Memory belongs to own cell

β Step (even iterations)
- Performed during even iterations
- Two streamings and one collision
- Memory belongs to neighbors
Single GPU optimization: The A-A Pattern

Only one memory layout is used for all iterations. This saves memory by a factor of two ⇒ twice as many cells per GPU

\[\alpha\] Step (odd iterations)
- Performed during odd iterations
- Only one collision is performed
- Memory belongs to own cell

\[\beta\] Step (even iterations)
- Performed during even iterations
- Two streamings and one collision
- Memory belongs to neighbors

Data is always read and written to the same memory location.
OpenCL Features

Command Queue

- Contains a list of commands
- Commands can be executed in-order or out-of-order

Event

- Can be used to determine the state of a command
- Mechanism to implement synchronization between commands
- Complex dependency graphs can be realized by dependencies of events
OpenCL Features

Command Queue
- Contains a list of commands
- Commands can be executed in-order or out-of-order

Event
- Can be used to determine the state of a command
- Mechanism to implement synchronization between commands
- Complex dependency graphs can be realized by dependencies of events

Only the concurrent execution of several command queues enables the parallel execution of several commands!
Single Boundary Kernel - Single Command Queue (SBK-SCQ) 1/2

MPI_Isend()
MPI_Irecv()
GPU
CPU
communication (GPU ↔ CPU)
communication (CPU ↔ CPU)

compute halo cells
compute inner cells

computations done by GPU
computations done by CPU
communication
command queue
idle times
Single Boundary Kernel - Single Command Queue (SBK-SCQ) 2/2

communication (CPU ↔ CPU)

communication (GPU ↔ CPU)

one single command queue

one simple kernel for halo cells

one simple kernel for halo cells

computations done by CPU

computations done by GPU

MPI_Isend()

MPI_Irecv()

GPU

CPU

X₀ Xₙ Y₀ Yₙ Z₀ Zₙ

X₀ Xₙ Y₀ Yₙ Z₀ Zₙ

X₀ Xₙ Y₀ Yₙ Z₀ Zₙ

X₀ Xₙ Y₀ Yₙ Z₀ Zₙ

compute halo cells

compute inner cells

idle times

Christoph Riesinger: High Scalability of Lattice Boltzmann Simulations with Turbulence Models using Heterogeneous Clusters
SIAM PP’2014, February 20, 2014
Multi Boundary Kernels - Single Command Queue (MBK-SCQ) 1/2

communication (CPU ↔ CPU)

communication (GPU ↔ CPU)

compute halo cells
compute inner cells

MPI_Isend()
MPI_Irecv()

multiple kernels for halo cells

computations done by CPU
computations done by GPU

Christoph Riesinger: High Scalability of Lattice Boltzmann Simulations with Turbulence Models using Heterogeneous Clusters
SIAM PP’2014, February 20, 2014
MPI_Isend()  
MPI_Irecv()  

communication (CPU ↔ CPU)  
communication (GPU ↔ CPU)  

one single command queue  
compute halo cells  
compute inner cells  

multiple kernels for halo cells  
computations done by GPU  
computations done by CPU  
command queue  
idle times  

Christoph Riesinger: High Scalability of Lattice Boltzmann Simulations with Turbulence Models using Heterogeneous Clusters  
SIAM PP’2014, February 20, 2014
Multi Boundary Kernels - Multi Command Queues (MBK-SCQ)

CPU

MPI_Isend()

communication
(GPU ↔ CPU)

MPI_Irecv()

multiple command queues for halo cells

GPU

compute halo cells

compute inner cells

compute halo cells

X0 Xn Y0 Yn Z0 Zn

X'0 X'n Y'0 Y'n Z'0 Z'n

multiple kernels for halo cells

idle times

computations done by GPU

computations done by CPU

communication

command queue
Topics

Motivation

The Lattice Boltzmann Method

Optimization

Single GPU optimization: A-B vs. A-A Pattern
Multi GPU optimization: Communication Schemes

Results

Runtime
Lattice Updates
Parallel Efficiency

Conclusion & Outlook
## Test Configuration

**Tödi Cluster in Switzerland**

**CPUs:** AMD Opteron Interlagos 6272  
- number of cores per CPU: 16  
- total number of CPUs: 272  
- total number of CPU cores: 4352

**GPUs:** NVIDIA Tesla K20x  
- number of CUDA cores per GPU: 2688  
- total number of GPUs: 272 (256 used)  
- total number of CUDA cores: 731,136
Runtime (Weak Scaling)

Optimal Runtime
Basic
SBC-SCQ
MBC-SCQ
MBC-MCQ

Number of GPUs (logarithmic scale)
Runtime (Weak Scaling)
Optimal Runtime
Basic
SBC-SCQ
MBC-SCQ
MBC-MCQ

normalized Runtime (linear scale)

0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0

Normalized Runtime (linear scale)

Optimal Runtime
Basic
SBC-SCQ
MBC-SCQ
MBC-MCQ

Number of GPUs (logarithmic scale)
Lattice Updates (Weak Scaling)

GLUPS (logarithmic)

Number of GPUs (logarithmic)
Parallel Efficiency (Weak Scaling)

Christoph Riesinger: High Scalability of Lattice Boltzmann Simulations with Turbulence Models using Heterogeneous Clusters
SIAM PP’2014, February 20, 2014
Topics

Motivation

The Lattice Boltzmann Method

Optimization
  Single GPU optimization: A-B vs. A-A Pattern
  Multi GPU optimization: Communication Schemes

Results
  Runtime
  Lattice Updates
  Parallel Efficiency

Conclusion & Outlook
Conclusion & Outlook

Conclusion

- Biggest gain in performance is achieved by using SBK-SCQ instead of a basic scheme
- More sophisticated schemes (MBK-MCQ) achieve results similar to less sophisticated schemes (SBK-SCQ) ⇒ Looks like NVIDIA OpenCL does not support overlapping of command queues
- Degree of parallel efficiency decreases linearly with number of GPUs

Outlook

- Improve poor parallel efficiency (∼40%), by running multiple command queues in parallel
- By considering the network topology
- Exploit CPUs not just for communication but also for computation
Conclusion & Outlook

Conclusion

- Biggest gain in performance is achieved by using SBK-SCQ instead of a basic scheme.
- More sophisticated schemes (MBK-MCQ) achieve results similar to less sophisticated schemes (SBK-SCQ).
  ⇒ Looks like NVIDIA OpenCL does not support overlapping of command queues.
- Degree of parallel efficiency decreases linearly with number of GPUs.

Outlook

- Improve poor parallel efficiency (~ 40%),
  - by running multiple command queues in parallel
  - by considering the network topology
- Exploit CPUs not just for communication but also for computation.
Final slide