Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement

SIAM PP 2014, Portland

February 19, 2014

Florian Schornbaum, Ehsan Fattahi, David Staubach, Christian Godenschwager, Martin Bauer, Ulrich Rüde

Chair for System Simulation
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
Outline

• Introduction
  • The *walBerla* Simulation Framework
  • The Lattice Boltzmann Method

• Uniform Grids
  • Domain Decomposition & Parallelization
  • Performance / Benchmarks

• Statically Refined Grids
  • Lattice Boltzmann & Grid Refinement
  • Domain Decomposition & Load Balancing
  • Performance / Benchmarks

• Outlook on Dynamic Grid Refinement / Conclusion
Introduction

- The *walBerla* Simulation Framework
- The Lattice Boltzmann Method
Introduction

• **walBerla** (widely applicable Lattice Boltzmann framework from Erlangen):
  • main focus on **CFD** (computational fluid dynamics) simulations based on the **lattice Boltzmann method** (LBM)
  • at its very core designed as an **HPC** software framework:
    • scales from laptops to current petascale supercomputers
    • largest simulation: 1,835,008 processes (IBM Blue Gene/Q @ Jülich)
    • **hybrid parallelization**: MPI + OpenMP
    • **vectorization** of compute kernels
  • written in **C++(11)**
  • support for different platforms (Linux, Windows) and compilers (GCC, Intel XE, Visual Studio, llvm/clang, IBM XL)
  • coupling with in-house rigid body physics engine *pe*
  • **open source** → [http://www.walberla.net](http://www.walberla.net)

Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement
Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
Introduction

• The lattice Boltzmann method:
  • regular grid with multiple particle distribution functions (= scalar values) per cell [D2Q9, D3Q19, D3Q27, ...]
  • explicit method → time stepping
  • two steps: stream (neighbors) & collide (cell-local)

• For the collision, different operators exist: SRT, TRT, MRT.
• Macroscopic quantities (velocity, density, ...) can be calculated from the particle distribution functions.
Uniform Grids

- Domain Decomposition & Parallelization
- Performance / Benchmarks
Uniform Grids

- **Domain Decomposition:**
  - regular decomposition into blocks containing uniform grids

- **Parallelization:**
  - data exchange on borders between blocks via ghost layers

  [special case of our much more general forest of octrees data structure → non-uniform/refined grids]
Uniform Grids

Load balancing can be based on either space-filling curves (Z-order/Morton order, Hilbert curve) using the underlying forest of octrees or graph partitioning (METIS, ...). Whatever fits best the needs of the simulation.
Uniform Grids

geometry given by surface mesh

load balancing

domain decomposition into blocks

empty blocks are discarded

Load balancing can be based on either space-filling curves (Z-order/Morton order, Hilbert curve) using the underlying forest of octrees or graph partitioning (METIS, ...).

Whatever fits best the needs of the simulation.
Uniform Grids

geometry given by surface mesh

domain decomposition into blocks

load balancing

empty blocks are discarded

allocation of block data (→ grids)

The domain decomposition and load balancing can be performed during the actual simulation ... OR ...
Uniform Grids

- Geometry given by surface mesh
- Domain decomposition into blocks
- Load balancing
- Empty blocks are discarded
- Separation of domain partitioning from simulation
- File size: kilobytes to few megabytes
- Allocation of block data (→ grids)

Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement
Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
Uniform Grids

All of this (the entire pipeline) works just the same when grid refinement is used.
Uniform Grids - Performance

- Benchmark Environments:
  - JUQUEEN (TOP500: 8)
    - Blue Gene/Q, 459K cores, 1 GB/core
    - compiler: IBM XL / IBM MPI
  - SuperMUC (TOP500: 10)
    - Intel Xeon, 147K cores, 2 GB/core
    - compiler: Intel XE / IBM MPI

- Benchmarks (LBM D3Q19):
  - lid-driven cavity
    - weak scaling
      (= const. number of cells per core)
  - coronary artery tree
    - strong scaling
      (= const. total number of cells)


Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement

Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
Uniform Grids - Performance

- SuperMUC – single node (3.34 million cells per core)

⇒ LBM: low FLOPs to bytes ratio → memory intensive
• SuperMUC – single node (3.34 million cells per core)

⇒ limited by memory bandwidth
Uniform Grids - Performance

- JUQUEEN – single node (1.73 million cells per core)

$\Rightarrow$ limited by memory bandwidth

vectorized compute kernel

already quite optimized!

naïve, straightforward implementation

SRT

TRT

MLUP/s

cores

bandwidth limit

1 2 4 8 16

cores

Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement

Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
Uniform Grids - Performance

- SuperMUC – TRT kernel (3.34 million cells per core)

0.99 x 10^{12} cells updated per second!

(19 values per cell)

Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement
Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
Uniform Grids - Performance

- JUQUEEN – TRT kernel (1.73 million cells per core)

\[ 2.1 \times 10^{12} \text{ cells updated per second!} \]
\[ (19 \text{ values per cell}) \]
\[ \Rightarrow 40 \times 10^{12} \text{ values updated per second!} \]
\[ \Rightarrow 0.41 \text{ PFlop/s} \ (0.41 \times 10^{15} \text{ Flop/s}) \]
• SuperMUC – TRT kernel (2.1 million fluid cells)

"extreme" strong scaling
more processes = shorter time to solution!
• JUQUEEN – TRT kernel (16.9 million fluid cells)

“extreme” strong scaling
more processes = shorter time to solution!
Statically Refined Grids

- Lattice Boltzmann & Grid Refinement
- Domain Decomposition & Load Balancing
- Performance / Benchmarks
Statically Refined Grids

- Lattice Boltzmann & Grid Refinement:
  - Almost all grid refinement schemes for the lattice Boltzmann method rely on a **2:1 balance** between neighboring cells:

- **waLBerla** now uses a massively parallel implementation of the refinement scheme presented in [1] (also relies on 2:1 balance)
- consequences of the 2:1 balance:
  - **twice as many time steps on the fine grid** as on the next coarser grid
  - In 3D, for each finer grid level, the memory requirement increases by a factor of 8 and the generated workload by a factor of 16.

Domain Decomposition:

- "blocks containing regular grids distributed among all processes"
- \textit{distributed forest of octrees} (→ 2:1 balance)
- Each process only knows about its \textbf{own blocks} and their \textbf{neighbors}, but has no information about the \textbf{rest of the domain}.
- Perfectly distributed data structure that \textbf{scales to huge numbers of processes} without any runtime overhead

Property of this Distributed Data Structure:

- can be viewed as a graph: each block is connected to all of its neighboring blocks → enables all kinds of graph algorithms
• Comparison with Uniform LBM:
  • The **setup phase remains unchanged**: The decoupling (via a file) of the domain decomposition & initial load balancing from the actual simulation is still possible.
  • All the refinement “magic” happens during communication (fine and coarse blocks share ghost layer regions → refinement requires **multiple ghost layers** per block†).
  • **Time stepping becomes more complicated** (simple, uniform LBM: boundary handling → stream & collide → communication).
  • **All the compute kernels remain unchanged!**

† Even though for the computation/interpolation multiple ghost layers are required, the communication is heavily optimized and the average number of bytes communicated per block only slightly increases compared to a uniform simulation.
Statically Refined Grids

- **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

  Example:
  24 blocks on four different levels
  (forest with 3 trees)
Statically Refined Grids

- **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

Example:

24 blocks on four different levels (forest with 3 trees)
Load Balancing (adapted to our LBM refinement implementation):

- one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)

numbers represent corresponding grid levels
Statically Refined Grids

- **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)
Statically Refined Grids

- **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)
Statically Refined Grids

• **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)
Statically Refined Grids

• Load Balancing (adapted to our LBM refinement implementation):
  • one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)
Statically Refined Grids

- **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

Example:

24 blocks on four different levels (forest with 3 trees)

- “even” distribution to all available processes

Example (cont.):

distribution to six available processes - level by level

process: 0 1 2 3 4 5
Load Balancing (adapted to our LBM refinement implementation):

- one space filling curve (Hilbert curve) per grid level

Example:
24 blocks on four different levels (forest with 3 trees)

```
0 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
```

“even” distribution to all available processes

Example (cont.):
Distribution to six available processes - level by level

```
process: 0 1 2 3 4 5
```
Statically Refined Grids

Load Balancing (adapted to our LBM refinement implementation):
- one space filling curve (Hilbert curve) per grid level

Example: 24 blocks on four different levels (forest with 3 trees)

"even" distribution to all available processes

Example (cont.): distribution to six available processes - level by level
• **Load Balancing** (adapted to our LBM refinement implementation):
  
  - one space filling curve (Hilbert curve) per grid level

  Example:
  
  24 blocks on four different levels (forest with 3 trees)

  • “even” distribution to all available processes

  Example (cont.): distribution to six available processes - level by level
• **Load Balancing** (adapted to our LBM refinement implementation):
  - one space filling curve (Hilbert curve) per grid level

\[
\begin{array}{ccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array}
\]

Example: 24 blocks on four different levels (forest with 3 trees)

- “even” distribution to all available processes

\[
\begin{array}{ccccccc}
1 & 1 & 1 & 1 & 1 & 0 & \\
2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 3 & 2 & 2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array}
\]

Example (cont.): distribution to six available processes - level by level

process: 0 1 2 3 4 5
Statically Refined Grids

• Benchmark Environments:
  • JUQUEEN (TOP500: 8)
    • Blue Gene/Q, 459K cores, 1 GB/core
    • compiler: IBM XL / IBM MPI
  • SuperMUC (TOP500: 10)
    • Intel Xeon, 147K cores, 2 GB/core
    • compiler: Intel XE / IBM MPI

• Benchmark (LBM D3Q19 TRT):
  • lid-driven cavity
  • weak scaling
  • 6.6875 blocks per process
  • 4 grid levels
  • finest level ...
    ... covers 1.4% of space
    ... generates 80% of workload
  • coarsest level ...
    ... covers 78% of space
    ... generates 1.1% of workload
Statically Refined Grids

• SuperMUC – TRT kernel (3.34 million cells per core)

Massively Parallel Lattice Boltzmann-based Simulations with Grid Refinement
Florian Schornbaum - FAU Erlangen-Nürnberg - February 19, 2014
- JUQUEEN – TRT kernel (1.81 million cells per core)
Statically Refined Grids

• LBM with refinement only achieves half the number of cell updates per second as regular LBM without refinement.

• Refinement Overhead:
  • much more complex communication patterns which additionally involve interpolation
  • more complex time stepping scheme
  • multiple blocks per process ⇔ smaller blocks

BUT

LBM with grid refinement requires much less memory and far fewer cells (100 times fewer ...)!
Statically Refined Grids

• Example Application: A Study of the Vocal Fold

⇒ http://youtu.be/kUPf__THVZs
Statically Refined Grids

• Example Application: A Study of the Vocal Fold

DNS (direct numerical simulation)
Reynolds number: 1000 / D3Q19 TRT
4300 processes @ SuperMUC
101,466,432 fluid cells
25,800 blocks with 16 x 16 x 16 cells
Statically Refined Grids

• Example Application: A Study of the Vocal Fold

number of different grid levels: 5
95.5 time steps / sec (finest grid)
total number of time steps (finest grid): 864,000
without refinement: 55.2 times more memory ...
... and 98.6 times the workload
Outlook on Dynamic Grid Refinement / Conclusion
• Dynamic Grid Refinement:
  • space filling curve? (see p4est and work done by C. Burstedde et al.)
  • multiple space filling curves – one for each grid level?
  • repartitioning based on graph algorithms?
    → diffusive algorithms! (see work of H. Meyerhenke et al.)

• Conclusion:
  • The walBerla framework now has a highly efficient, massively parallel implementation of LBM for ...
    • ... uniform grids as well as ...
    • ... statically refined grids.
  ⇒ next milestone: support for dynamic grid refinement / runtime adaptivity
THANK YOU FOR YOUR ATTENTION!

QUESTIONS?

(visit http://www.walberla.net)